Apple *II*

Applesoft/DOS Programmer's Tool Kit
Quick Reference Card

Applesoft Programmer's Assistant Command Summary

Commands work in immediate execution only. Type & plus first letter of each command (first two letters for &MAnual). Typing whole command is optional.

&Auto ⟨start⟩, ⟨inc⟩

Automatically assigns line numbers beginning with <start> at increments of <inc>. <Start> is required; default for <inc> is 10.

&Compress

Removes all nonreferenced REM statements. Contents of referenced REM statements are removed, but REM statements left intact.

&Hold

Stores RAM-resident program above HIMEM: until retrieved via `&Merge`.

&Keys

Allows display of certain characters unaccessible from Apple II and Apple II Plus keyboard:

To Get This	Type This
_	CONTROL-O
\	CONTROL-L
[	CONTROL-K

&Length

Gives byte length of program in memory.

&MAnual

Turns off automatic line numbering (`&A`), extra characters (`&K`).

&Merge

Combines program stored by `&Hold` with current RAM-resident program.

&Noshow

Turns off `&Show`.

&Renumber ⟨start⟩, ⟨inc⟩, ⟨first⟩, ⟨last⟩

Renumbers program beginning at <start> in increments of <inc>. To renumber part of a program, also state lowest <first>, highest <last> line to renumber. Defaults are <100>, <10>, <lowest>, <highest>.

&Show

Displays embedded control characters in highlighted text (II and II plus only).

&Xref

Lists each variable and all line numbers in which each appears.

Boston Window Command Summary

Functional Command Summary

Editor Access

Enter editor	CONTROL-E
Quit editor	CONTROL-Q

Cursor Movement

Open window	CONTROL-O
Move left one character	← (IIe) or ESC J or CONTROL-A
Move right one character	→ (IIe) or ESC K or CONTROL-S
Move up one line	↑ (IIe) or ESC I or CONTROL-W or -K
Move down one line	↓ (IIe) or ESC M or CONTROL-Z or -J
Move to left edge of screen	CONTROL-B
Move to right edge of screen	CONTROL-E
Scroll toward top of program	CONTROL-T
Scroll toward bottom of program	CONTROL-V

Delete

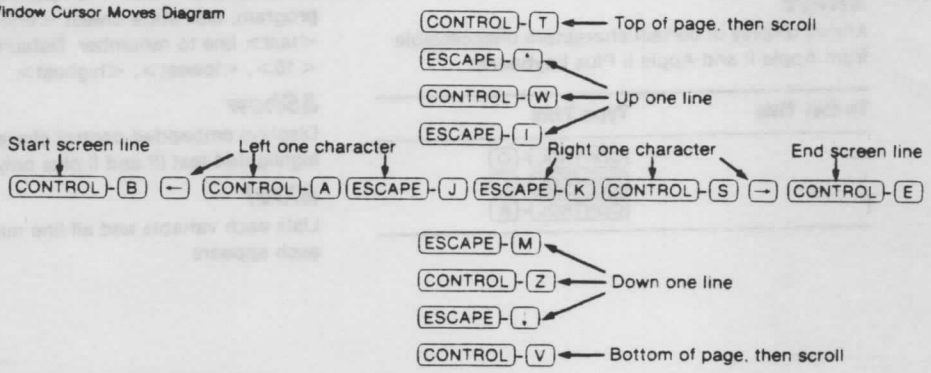
Delete character left	CONTROL-H (with ☐ on IIe) or ← (with ☐ on IIe)
Delete character at cursor	CONTROL-D
Delete program line	CONTROL-L
Delete to end of program line	CONTROL-Y
Restore last character deleted	CONTROL-U (with ☐ on IIe) or → (with ☐ on IIe)

Apple IIe DELETE key deletes character left without putting it into buffer. Restore command cannot recover characters removed via DELETE.

Copy

Open, close copy buffer	CONTROL-C
Insert copy buffer contents	CONTROL-I or TAB (IIe)

Boston Window Cursor Moves Diagram



Find, Replace

Find specified string	CONTROL-F
Add wildcard character	CONTROL-W
Replace string with specified string	CONTROL-R
Global Replace (after CONTROL-F CONTROL-R)	CONTROL-G

Start New Line

Start new higher line	CONTROL-M or RETURN
Start new lower line	CONTROL-^

Miscellaneous Functions

Autonumbering ON-OFF	CONTROL-N
Autoinsert ON-OFF	CONTROL-@
GOTO specified line number	CONTROL-G
Insert control character	CONTROL-P

Alphabetic Command Summary

CONTROL-A Moves cursor left one character —same as ESC J

CONTROL-B Moves cursor to left edge of current display line

CONTROL-C Opens/closes copy buffer

CONTROL-D Deletes character at cursor position, moves all text left one space

CONTROL-E From Applesoft, enters editor; from editor, moves cursor to right edge of current screen line

CONTROL-F FINDs specified string

CONTROL-G GOTO specified line number; at end of CONTROL-F CONTROL-R sequence, does Global Replace

CONTROL-H (with ☐ on IIe) Deletes character to left of cursor, moves all text left one space—same as ☐-←

CONTROL-I Inserts contents of copy buffer at cursor position

CONTROL-J Moves cursor down one line—same as CONTROL-Z, ESC M

CONTROL-K Moves cursor up one line—same as CONTROL-W, ESC I

CONTROL-L Deletes program line containing cursor

CONTROL-M Opens window on current program line, closes window on new line; if autonumbering is in effect, assigns line number to new line higher than current line —same as RETURN key press

CONTROL-N If autonumbering is ON, turns OFF; if autonumbering is OFF, turns ON

CONTROL-O Opens window

CONTROL-P Use before adding control character to text; control character displayed in inverse video

CONTROL-Q Leaves the editor and returns to Applesoft with cursor at bottom left of screen

CONTROL-R Used after CONTROL-F to establish replacement string or to effect the replacement

CONTROL-S Moves cursor right one character—same as ESC K

CONTROL-T Moves cursor toward top of window or scrolls to start of program; stopped by press of any character key

CONTROL-U (with ☐ on the IIe) Restores one character from delete buffer—same as ☐-→

CONTROL-V Moves cursor toward bottom of window or scrolls to end of program; stopped by press of any character key

CONTROL-W Moves cursor up one line—same as CONTROL-K, ESC I; wildcard character in FIND string

CONTROL-X Cancels current command (FIND or GOTO)

CONTROL-Y Deletes into buffer all characters from cursor to end of program line

CONTROL-Z Moves cursor down one line—same as CONTROL-J, ESC M

CONTROL-@ If autoinsert is ON, turns OFF; if autoinsert is OFF, turns ON

CONTROL-^ Opens window on current line; closes window on line to have lower line number than current line; if autonumbering is in effect, assigns line number to new line lower than current line

CONTROL-H (with ☐ on IIe) or ← (with ☐ on IIe)
CONTROL-D
CONTROL-L
CONTROL-Y
CONTROL-U (with ☐ on IIe) or → (with ☐ on IIe)

acter left without
nand cannot
ELETE.

CONTROL-C
CONTROL-I
or TAB (IIe)

d screen line
ONTROL-E

croll

CONTROL-M
or RETURN
CONTROL-^

CONTROL-N
CONTROL-@
CONTROL-G
CONTROL-P

Special CALLs and POKEs

Reconnect Boston Window, full screen	CALL 34698
Reconnect Boston Window, half screen	CALL 34689
Set AUTONUM to stipulated increment	POKE 38925, <n>
Delete nonquoted spaces (buffer)	POKE 37048, 160
Restore to normal	POKE 37048, 0
Delete nonquoted spaces (screen)	POKE 37789, 186
Restore to normal	POKE 37789, 0
Display PRINT as ?	POKE 37781, 186
Restore to normal	POKE 37781, 0
Turn off status line	POKE 38638, 96
Display status line	POKE 38638, 152

Functions of Special Keys

DELETE	(On IIe only) Deletes character left (does *not* go into buffer)
ESC	Turns on escape mode
RETURN	Opens window on curent line, inserts blank line below current line, and closes editing window around blank line
↑	(On IIe only) Moves cursor up one screen line
↓	(On IIe only) Moves cursor down one screen line
←	Apple II and II Plus: Removes character left and puts into delete buffer
	Apple IIe: Moves cursor left one character; with ☐, removes character left and puts into delete buffer
→	Apple II and II Plus: Restores last character put into delete buffer
	Apple IIe: Moves cursor right one character; with ☐, restores last character put into delete buffer
TAB	(On IIe only) Inserts copy buffer into text—same as CONTROL-I

High Resolution Character Generator Summary

Functional Command Summary

Display Characteristics		
Inverse video	CONTROL-I	replaces Applesoft INVERSE command
Normal video	CONTROL-N	replaces Applesoft NORMAL command
All uppercase	CONTROL-K	K for Kaps Lock
All lowercase	CONTROL-L	
Shift	CONTROL-S	Next character upper, all after lower
Choose character set	CONTROL-A <n>	
Clearing the Screen		
Clear page	CONTROL-P	
Clear to end of line	CONTROL-E	
Clear to end of page	CONTROL-F	
Choosing Display Page		
Page 1 as primary page	CONTROL-O CONTROL-A	
Page 2 as primary page	CONTROL-O CONTROL-B	
Display primary page	CONTROL-O CONTROL-D	

Page Scrolling

Scroll Text window	CONTROL-O CONTROL-S	
Wrap Text window	CONTROL-O CONTROL-W	

Overlays

Text Overstrike	CONTROL-O CONTROL-O	Old dots combine with new dots on same page
Complement Overlay	CONTROL-O CONTROL-C	New dots covering old dots on same page cause color of old dots to be complemented
Transparent Overlay	CONTROL-O CONTROL-T	New dots overlay character on secondary display; result transferred to primary
Reverse Overlay	CONTROL-O CONTROL-R	Same as Transparent, except overlaid dots on secondary display complemented
Print Replace	CONTROL-O CONTROL-P	The standard condition

Windows

Full-screen window	CONTROL-Y
Upper left corner window	CONTROL-V
Lower right corner window	CONTROL-W

Block Display

Start Block Display	CONTROL-B	All characters after this command taken as single unit until CONTROL-D occurs
End Block Display	CONTROL-D	defines end of block; each character typed after this taken as individual unit again

Accessing Machine Language Routines

Call user routine 1	CONTROL-O CONTROL-Y
Call user routine 2	CONTROL-O CONTROL-Z

Issuing the first of these commands does a JSR to relocatable location 10 in HRCG (or location 13 in the case of the second command). There HRCG expects to find your subroutine's address in locations 10 and 11 (or 13 and 14 for routine 2), low byte first.

Original Parameters

Restore parameters CONTROL-Z

Clearing HRCG

Turn off HRCG CONTROL-RESET FF

Alphabetic Command Summary

Default parameters are marked with an asterisk (*).

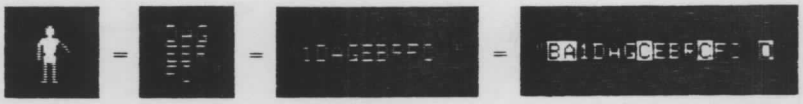
CONTROL-A <n>	Selects character set n
CONTROL-B	Begins Block Display
CONTROL-C	Carriage return
CONTROL-D	Delimits Block Display
CONTROL-E	Clears to end of line
CONTROL-F	Clears to end of screen
CONTROL-I	Inverse video
CONTROL-K	*Caps lock
CONTROL-L	Lowercase
CONTROL-N	*Normal video
CONTROL-O	Selects option (see list below)
CONTROL-P	Clears page
CONTROL-Q	Homes cursor
CONTROL-S	Shift
CONTROL-V	Sets text window (upper-left)
CONTROL-W	Sets text window (lower-right)
CONTROL-Y	*Sets text window (full screen)
CONTROL-Z	Restores default parameters

CONTROL-O CONTROL-A	*Primary = page 1
CONTROL-O CONTROL-B	Primary = page 2
CONTROL-O CONTROL-C	Complement Overlay
CONTROL-O CONTROL-D	Display primary
CONTROL-O CONTROL-O	Text Overstrike
CONTROL-O CONTROL-P	*Print Replace
CONTROL-O CONTROL-R	Reverse Overlay
CONTROL-O CONTROL-S	*Scroll Text
CONTROL-O CONTROL-T	Transparent Overlay
CONTROL-O CONTROL-W	Wrap Text
CONTROL-O CONTROL-Y	Call user routine 1
CONTROL-O CONTROL-Z	Call user routine 2

Block Character Forms

Below are the four ways you might see the same characters appear on the display. The first figure is seen through HRCG, the second in HRCG after pressing CONTROL-RESET, the third in a normal line listing, and the fourth in a line listing with control characters visible.

Control characters highlighted



Complementary Colors

Black . . White
Blue . . . Orange
Green . . Violet

Table of Contents

Introduction

Welcome to the Applesoft/DOS Tool Kit, part of the DOS Programmer's Tool Kit package. This introduction briefly describes the capabilities and requirements of the Applesoft/DOS Tool Kit, and gives you a look at how this manual is set up.

About the Applesoft/DOS Tool Kit

The Applesoft/DOS Tool Kit is designed to help you use the full programming capabilites of your Apple II series computer. Here you'll find a collection of software tools to aid you in creating, developing, and modifying both text and graphics-based Applesoft programs.

The programs that make up the Applesoft/DOS Tool Kit are all to be found on the disk labeled DOS Programmer's Tool Kit Volume I.

Who Can Use the Tool Kit?

This kit is intended for experienced programmers (or those aspiring to programmerhood) who are familiar with Applesoft and DOS 3.3. It is not mainly intended for the novice, although some programs can be used by novices. The Ribbit and Skylab games can be played by children. Before you use this tool kit, you should be familiar with both DOS and Applesoft.

Equipment You'll Need

To use all the programs in the tool kit, you should have

- an Apple II series computer with at least 48K of RAM
- at least one disk drive (16-sector)
- a DOS 3.3 SYSTEM MASTER disk
- a set of hand controls

The controls are necessary for Animatrix (the character graphics editor) and the games.

About the Manual

This manual explains how to use the programs in the Applesoft/DOS Tool Kit. You don't need to read it from beginning to end, but can pick and choose the chapters that describe capabilities you're interested in learning about and using.

How the Manual Is Organized

The manual is divided into three parts.

Chapters 1 through 4 describe utilities useful for all programmers. These chapters include introductions to and tutorials for the Applesoft Programmer's Assistant, Boston Window, System Identification Routines, and From Machine Language to BASIC. There are two versions of Chapter 2 (Boston Window)—one for people with Apple II or II Plus computers, and one for folks with Apple IIe machines.

Chapters 5 through 7 are concerned with the tools for developing and using special text and animation character sets; they cover Animatrix, High Resolution Character Generator, and the Relocating Loader.

Chapter 8, Bringing It All Together, uses most of the programs in the tool kit to show how to implement character animation in any program.

Here's a list of the programs discussed in this manual, and a brief description of each:

Part 1—For All Programmers

Chapter 1	Applesoft Programmer's Assistant—a utility that adds 11 new commands to Applesoft, including ones for renumbering and merging programs
Chapter 2 (and 2e)	Boston Window—an Applesoft program editor
Chapter 3	System Identification Routines—a brief set of routines you can add to your programs so they can distinguish among models and configurations of Apple II series computers
Chapter 4	From Machine Language to BASIC—a utility for converting machine language routines to BASIC program lines

Part 2—For High Resolution Graphics and Animation Buffs

Chapter 5 Animatrix—a graphics editor for creating high resolution character sets for special alphabets and animation sequences

Chapter 6 High Resolution Character Generator—a utility adding around 30 commands to Applesoft for the manipulation and presentation of character sets on the high resolution displays

Chapter 7 The Relocating Loader—a pair of programs (RBOOT and RLOAD) used to load and relocate special Rfiles above HIMEM:

Part 3—All Programmers Take Note

Chapter 8 This chapter ties together most of the programs discussed in the manual. Everyone who uses the tool kit should read it at least once. Programmers interested in the graphics utilities will find it especially useful.

This manual makes frequent reference to the game programs RIBBIT and SKYLAB, to the character animation demonstration program MAXWELL, and to various character sets. All of these programs and files are to be found on the disk labeled DOS Programmer's Tool Kit Volume I. Additionally, some of the chapters make reference to the 6502 Assembler manual and the EDASM program, also included in the DOS Programmer's Tool Kit package.

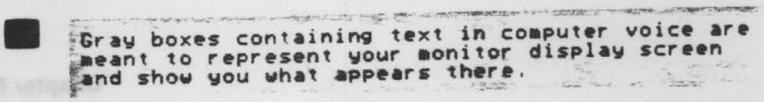
How the Manual Is Designed

Look for these visual aids throughout the manual:

Words with which you may be unfamiliar appear in **boldface**. They are defined in the glossary at the end of the manual.

Words you should type and words as they appear on the monitor display screen are printed in `computer voice`.

By the Way: Gray boxes contain incidental information and helpful hints.



Warning
Boxes like this indicate potential problems or disasters.

Notes in the margins are signposts that summarize main points so you can more easily flip to a particular discussion, example, or format.

For Further Information...

You can find more information on many of the concepts presented in these chapters in the Applesoft and DOS reference manuals, and in the Apple II series hardware reference manuals. Especially helpful are the appendixes in the *Applesoft BASIC Programmer's Reference Manual* that discuss high-resolution and text pages, and the structure of system memory.

Warning
Before you go any further, make a copy of the two DOS Programmer's Tool Kit disks. Use the COPY program on your DOS SYSTEM MASTER disk to do it. Then put away the originals. Do it now!

Applesoft Programmer's Assistant

Applesoft Programmer's Assistant

The Applesoft Programmer's Assistant (or APA for short) adds 11 new commands to Applesoft to help you write and change programs. APA can number lines automatically as you type in a program, renumber a program, merge one program with another, and eliminate REM statements to save program space. It can also cross-reference locations of variables, and tell you how much memory space your program occupies. And if you have an Apple II or II Plus (as opposed to a IIe), APA lets you type special characters usually not available from the keyboard.

APA and Boston Window, the Applesoft program editor described in the next chapter, create a wonderful environment in which to write programs. It's an excellent idea to have both of these **utilities** in memory whenever you write Applesoft programs.

To Use APA and Boston Window Together: To use the features of both programs at the same time, you need to run the Boston Window program before you start up APA. To do that, first type BRUN BOSTON WINDOW and press RETURN. Then go on reading this chapter.

Starting Up

Because the process of getting APA up and running will clear out memory, be sure to save any program you've been working on. Then insert DOS Programmer's Tool Kit Volume I into the appropriate disk drive and type RUN LOADAPA.

For Technical Types Only: The program LOADAPA calls a special sequence of machine language routines, known collectively as the Relocating Loader, to do its work. Briefly, LOADAPA takes APA and loads it (get it?) into the computer at the highest point available for program and variable space (that is, at HIMEM:). Then it resets HIMEM: to just below APA, so that APA can live in memory at the same time as the Applesoft program with which you are working.

For a detailed description of the Relocating Loader, see Chapter 7 of this manual.

When you press RETURN the drive whirs and zicks and the LOADAPA program loads, relocates, and initializes APA. You see this on the screen:

```
APPLESOFT PROGRAMMER'S ASSISTANT
VERSION 1.1
(C) COPYRIGHT 1979, 1983 APPLE COMPUTER

&RENUMBER <START>,<INC>,<FIRST>,<LAST>
&HOLD
&MERGE
&LENGTH
&COMPRESS
&AUTO <START>,<INC>
&MANUAL
&XREF

]
```

Figure 1-1. APA Opening Display

The Applesoft prompt appears in the lower left corner of the screen and, after you enter NEW to clear Applesoft's memory, you're ready to begin a new program or to continue working with one stored on a disk.

& begins all APA commands

Whenever you want to use one of APA's commands, type the ampersand (&) followed by the first letter or letters of the command word and any optional parameters. If you wish, you can type the whole command name. Spaces are optional. The ampersand tells Applesoft to interpret what immediately follows as an APA command.

APA's commands fall into four categories: numbering programs, merging modules, conserving space, and displaying program information.

Numbering Programs

Being able to manipulate line numbering is essential for making program changes. APA gives you several commands for keeping on top of the numbers.

Automatic Line Numbering

&A does automatic line numbering

The Auto command gives you automatic line numbering and saves you the drudgery of typing in line numbers yourself. To issue the command, you type &A. This command must always be followed by the line number you want to start with. Adding an increment number is optional, but if you do add one, you must separate the line number from the increment number with a comma.

This command begins by assuming you want the numbers to increase by 10 from one line to the next. To write a program starting with line number 20 with increments of 10, simply type `&A 20` and press RETURN. Press the SPACE bar and the line number `20` appears. Type your program line, press RETURN, and press the SPACE bar again: the next line number, number `30`, appears. To start (or continue) numbering from, say, line 150, type `&A 150`.

If you want an increment other than 10, add a comma (and a space if you wish—it's optional) after the starting number and type the increment you want. Thus if you want a program to start numbering at line 15 with increments of 25, type `&A 15, 25`.

If you want to leave a line number unused, press RETURN and then the SPACE bar; the next line number appears, and no statement is typed under the previous one.

The (CONTROL) key is labeled (CTRL) on the Apple II and II Plus

If you change your mind about a program line while you are typing it and want to cancel it, press (CONTROL)-(X); the line is canceled. Press the (SPACE) bar and the same number (but not the code you typed) reappears.

Note: If you have APA loaded in memory at the same time as Boston Window (which is usually an excellent practice), using APA's Auto command disconnects Boston Window. Boston Window has its own automatic numbering feature, which you should use instead. See Chapter 2 for a description of Boston Window.

&MA turns off autonumbering

To turn off automatic line numbering, type &MA for MAnual and press (RETURN). MAnual is the only APA command that requires more than one letter to execute. MAnual is in effect when you load APA; you have to ask for automatic line numbering to get it.

Renumbering Lines

&R renumbers a program

The Renumber command renumbers all or any part of your program. with any starting line number and increment. Renumbering can be extremely useful when there are no more whole numbers available between lines and you need to squeeze another line in. It's one thing if your program looks like this:

```
1050 REM THIS IS A SAMPLE LINE
1150 REM THIS IS THE NEXT LINE, SO LINE NUMBERS
     1051 TO 1149 ARE OPEN
```

You've got all the space in the world to add new program lines. But if your program looks like this:

```
1047 REM ASSUME THIS LINE IS INDISPENSABLE
1048 REM DITTO THIS LINE
1049 REM YOU NEED THIS LINE TOO
1050 REM THIS IS ALSO A LINE YOU CAN'T LIVE
     WITHOUT
1051 REM LET'S SEE YOU SQUEEZE IN LINE 1050.5!
1052 REM THE NEXT 25 CONSECUTIVE LINES ARE ALSO
     TAKEN
```

you're in a world of trouble if you need an additional line somewhere around 1050. That's where &R comes in.

&R's Default Values

If you execute the Renumber command without specifying any parameters (that is, if you type &R and press RETURN), then APA assumes that you want to use the following numbers:

```
&R 100, 10, 0, 63999
```

These are &R's default values. 100 designates the lowest line number to be assigned, 10 is the increment between numbers, 0 is the lowest possible line number to be renumbered, and no numbers above 63999 can be assigned to a line. This command renumbers the entire program in memory, assigning 100 to the first line and incrementing each line by 10.

Here's an example of how this renumbering works. Clear out program memory by typing NEW and pressing RETURN. Then type this short program and save it under the name SAMPLE (you'll be using it again in its pristine form later):

```
10 PRINT "S"
20 PRINT "A"
30 PRINT "M"
40 PRINT "P"
50 PRINT "L"
60 PRINT "E"
```

Now type &R and press RETURN. Then list the program. It now looks like this:

```
100 PRINT "S"
110 PRINT "A"
120 PRINT "M"
130 PRINT "P"
140 PRINT "L"
150 PRINT "E"
```

Specifying a New Increment

You can specify a different starting line number and increment by using this format:

```
&R <new first line number>, <increment between lines>
```

For example, &R 1000, 50 gives you

```
1000 PRINT "S"
1050 PRINT "A"
1100 PRINT "M"
1150 PRINT "P"
1200 PRINT "L"
1250 PRINT "E"
```

Partial Renumbering

If you want to renumber only part of a program, you can specify the first and last old line numbers to be changed, in this format:

```
&R <new first line number>, <increment>,
   <first number to change>, <last number to change>
```

If you specify in your sample program `&R 900, 10, 1000, 1100`, for example, only the line numbers between 1000 and 1100 inclusive will change:

①This was line #1000
②Used to be line #1050
③Formerly line #1100

Moving a Block of Code

You can also do clever things like moving segments of a program from one position to another. Let's say that lines 920 through 1200 form a unit you want to move to the end of the program, beginning at line 5000 with increments of 20 between lines. It's done with this command:

```
&R 5000, 20, 920, 1200
```

As you've probably guessed, 5000 is the new first line number, 20 is the new increment, 920 is the first old line number to change, and 1200 is the last old line number to change.

If you issue the command in your sample program, you'll end up with this:

These lines used to be between the current #910 and #1250

Notice that the block of newly renumbered lines has moved to its correct place in numerical order.

Now add the following REM statement to this program and save it under the name SAMPLE2:

```
0 REM THIS IS A MERGE SAMPLE
```

You'll use this program again to learn the Merge command, coming up in the next section.

What &R Won't Do

APA won't let you use Renumber to **interleave** lines, which occurs when a line range is renumbered such that its new numbers overlap the numbers of an existing line range. Use &R to spread out the interval between lines or to move blocks of code from one part of the program to another. If you want to interleave line numbers, use the Merge command, described in the next section.

The &R command will change references to line numbers in GOTO's, GOSUB's, ONERR's, and the like; but it will not change line references embedded in REM statements. It will, however, change the number of the line containing the REM statement.

▲ **Warning**

Sometimes, especially with longer programs, &R takes a little longer than you might like. It seldom takes longer than a few seconds; but if it does, just be patient. Resist the temptation to press CONTROL-RESET. Pressing CONTROL-RESET during the renumbering process will certainly stop the show, but it might also do unspeakable things to your code. Be sure you have a copy of your program stored safely on disk before you attempt a large-scale renumber, just to be on the safe side.

Merging Modules

How often in programming have you wished you had a tool to join together separate programs or program modules? You probably find yourself using the same routines over and over again to do similar tasks: routines to manage text display, routines to prompt users for certain kinds of input, and so on. APA's merging facilities let you write these routines once, store them on disk, and then call and use them in various programs as you need them.

Holding a Program

The Hold command stores the current program in memory above HIMEM: so it can't be erased, and lets you load another program into the area your program just left.

Make sure that SAMPLE2 is still in memory. If it isn't, reload it from disk. It should look like this:

```
0  REM THIS IS A MERGE SAMPLE
900 PRINT "S"
910 PRINT "A"
1250 PRINT "E"
5000 PRINT "M"
5020 PRINT "P"
5040 PRINT "L"
```

&H puts a program on hold

Now type &H. If you try to list the program now, you'll get nothing! The program hasn't been destroyed; it's just been moved in memory. To bring it back, read on.

Merging Two Programs

Showing you how Merge works is easier than describing it. Load the program SAMPLE from disk, and list it. It should look like this:

```
10 PRINT "S"
20 PRINT "A"
30 PRINT "M"
40 PRINT "P"
50 PRINT "L"
60 PRINT "E"
```

&M merges two programs

Now type the command &M and list the program. You should get this:

```
0  REM THIS IS A MERGE SAMPLE
10 PRINT "S"
20 PRINT "A"
30 PRINT "M"
40 PRINT "P"
50 PRINT "L"
60 PRINT "E"
900 PRINT "S"
910 PRINT "A"
1250 PRINT "E"
5000 PRINT "M"
5020 PRINT "P"
5040 PRINT "L"
```

The program you had on hold has been called back and merged with the program you loaded from disk. Note that the lines have been interleaved; line 0 is where it belongs, before line 10. While interleaving can't happen when you use the &R command, &M does it quite nicely.

Warning

If the program on hold has line numbers that are the same as the program in active memory, APA gives you the message DUPLICATE LINE NUMBER and lists the offending lines. Then APA asks you if you want to continue. If you give it the go-ahead, the program lines in the hold area replace the similarly-numbered lines in active memory. If you tell APA not to continue, then you can change the program in memory before going on with the merge.

You can leave a program on hold indefinitely while you play around in active memory; but the world being as strange as it is, you really ought to make sure you have a copy stored on disk of both the program on hold and the program in active memory.

Dumping a File on Hold

Sometimes you'll have a program on hold that you later decide you don't want to merge with anything. To clear out the holding area and to reset HIMEM: to its appropriate place, type &M and then NEW.

Related Error Messages

If you try to issue Hold when there's no program in memory, APA delivers the message NO PROGRAM. If you try to issue Merge when there's no program in the holding area, APA displays NO HOLD FILE.

If the combined space taken by the program in the hold area and the program you are trying to load into active memory exceed available space, you get a PROGRAM TOO LARGE message. Besides the obvious solutions (like shortening one of the programs), there are two things you can try. If you have both APA and Boston Window loaded, temporarily remove Boston Window by shutting off your computer and restarting it. Then run APA and try again.

If there is still not enough room, or if Boston Window wasn't in memory to begin with, then you'll have to try to compress your programs—which happens to be the subject of the very next section.

Conserving Space

While BASIC's REM statements are extremely valuable for documenting a program as you write it, they do take up space. When you get into a space crunch, as described in the last few paragraphs of the previous section, you might find it necessary to remove REM statements from a program. Going through a program line by line looking for REM statements can be a tedious process; so APA provides a command to remove REM statements en masse.

&C takes out REM statements

After you've made certain you've saved a copy of the program with the REM statements still intact, type the command &C, for Compress. APA removes all REM statements from your program, and notifies you in both decimal and hexadecimal of how many bytes you've saved. Then you can save the condensed version under a different name from the documented one. Later if you revise the program, you can make your changes to the documented version and make a new compressed version of it.

Only nonreferenced REM statements are entirely removed; that is, if there is a GOTO, GOSUB, or other branch instruction to a line number that begins with REM, then that line number and the word REM is left intact. The content of the REM statement, however, is dropped.

Displaying Program Information

APA contains several commands for displaying program information that is usually difficult to get. Such information includes the length of the program, what control characters are embedded in its text, and the names and locations of variables.

Finding Out Program Length

&L shows length of program

To obtain the length of the program in memory, type &L, for Length. APA displays the program's byte length in both hexadecimal and decimal.

Displaying Control Characters

&S shows control characters

To make embedded control characters in your program visible, type &S, for Show. When you list your program, any embedded control characters are displayed in highlighted text. Show also lets you see any embedded control characters in a disk catalog or in any text file.

Apple IIe Users Take Note: The Show command won't work on your computer. To see embedded control characters, use the Boston Window (see Chapter 2e).

Boston Window Fans Take Note: Show duplicates the automatic control display function built into Boston Window (described in Chapter 2). If you have Boston Window loaded into memory at the same time as APA you needn't bother with Show. It won't hurt to use it, but you'll get an annoying flashing at the bottom of your screen, and in half-screen mode the screen will soon unattractively fill with discarded lines. And that will surely offend your sense of the aesthetic. To cancel the effects of Show on Boston Window, issue a CALL 34698.

&N makes control characters invisible

To turn Show off and to make embedded control characters invisible again, type &N, for Noshow. Control characters are no longer displayed.

Listing Variable Names and Locations

One of programming's most common bugs is the use of the same variable name to represent two (or more!) different variables. Xref lets you see what variable names are in use and where they are being used in a program.

&X cross-references variables

The Xref command gives you a variable cross-reference listing of your Applesoft program, listing each variable and the numbers of the lines in which the variable appears.

To see this work, load the RIBBIT program. Then type &X. After a few seconds (patience—RIBBIT's a long program), APA displays the cross-referenced list of RIBBIT's variables. Be ready to use CONTROL-S to stop the display of references; there are lots of variables in RIBBIT.

Xref shows only the first two characters in the variable name, no matter how long it is. That's because Applesoft distinguishes between variable names of the same type by looking at the first two characters and ignoring the rest.

In addition to the first two characters of each variable name, &X also displays % if the variable is an integer, $ if it's a string, and (if it's an array. If the variable is an integer array, then &X follows the variable name with %(. If it's a string array, APA adds $(.

Use Your Printer: If you have a printer, it makes sense to turn it on and direct the computer's output to it before you use Xref; that way you'll have a hard-copy list of the cross-references to work with.

Bonus for Apple II and II Plus Owners: Extra Characters

&K gets extra characters

Owners of the Apple II and Apple II Plus have had to struggle along without being able to directly type certain characters on the keyboard. APA adds a bonus for these folks—keyboard access to the underscore (_), backslash (\), and the left bracket ([ ).

When you issue the command &K, for Keys, and press the following control codes, these extra characters appear:

To Get This	Enter This
_	CONTROL-O
\	CONTROL-L
[	CONTROL-K

(By the way, although you might not know it, you have always been able to get the right bracket by typing SHIFT-M).

To cancel Keys, type &MA for MAnual.

No Free Lunch Department: You can't use both Keys and Auto at the same time. If you issue &K when Auto is on, you will turn Auto off and Keys on. If you issue &A when Keys is on, you will turn Keys off and Auto on. The solution, of course, is either to:

1. Sell your old Apple and buy a IIe, or
2. Have Boston Window and APA in memory at the same time and use Boston Window's automatic numbering function instead.

Both of these commands disconnect Boston Window, described in Chapter 2. To reconnect Boston Window, issue a CALL 34698.

Life is so simple when you know its little secrets.

Boston Window for the Apple II

Boston Window for the Apple II

The Boston Window is a powerful Applesoft BASIC program editor. It's like a word processor for programs, allowing you to type and modify Applesoft programs quickly and easily.

Using this editor you can, among other things, scroll both backward and forward through a program, copy all or part of a program line, insert or delete characters within a line, and search for and change statements and text strings either once or throughout your program. Coupled with Applesoft Programmer's Assistant (see Chapter 1), Boston Window creates a wonderfully simple and complete environment in which to program.

To Use Boston Window and APA Together: To use the features of both programs at the same time, you need to run the Boston Window Program before you start up APA. To do that, first type BRUN BOSTON WINDOW and press (RETURN). Then type RUN LOADAPA and press (RETURN).

Boston Window is designed for people with some Applesoft programming experience; this chapter assumes that you have been programming in Applesoft for a while. As with any complex program with a lot of features, there's a learning curve with Boston Window—it will take you a while to learn to use everything it has to offer. But it will be worth it!

Only Apple II Users Should Read This Chapter: This chapter has been written for people using Apple II computers. If you have an Apple IIe, turn to the chapter called (sensibly enough) Chapter 2e.

How This Chapter Is Organized

This chapter has two sections. Section 1 is a tutorial and should be read while you're sitting at your computer; if you follow it straight through to the end, you'll be walked through most of Boston Window's commands and options. If you're a first-time user of the program, we recommend you go through this section at least once to get the hang of using Boston Window.

Section 2 is a reference section; it contains summary tables of all commands and options, a chart of the cursor movement keys, and miscellaneous technical data.

Where The Name Comes From: The *Boston* part of the program's name comes, naturally enough, from the town where its author—Kenneth A. Tepper, Ph.D.—lives (an obscure suburb of New York, some 218 miles to its northeast; the town is rumored to have had some significance during the Revolutionary War). *Window* refers to an invisible frame that surrounds each program line while you're in the process of editing it. There'll be more about the window concept as we go along.

Section 1

Boston Window Tutorial

This tutorial assumes you are already familiar with Applesoft; it makes no attempt to teach the basics of programming. If you have no experience with Applesoft, spend some time with the *Applesoft Tutorial* manual and get the elementary stuff out of the way. Then come back here; we'll wait for you.

Starting Boston Window

Getting Boston Window started is really simple. Just insert the DOS Programmer's Tool Kit Volume I disk into the appropriate disk drive and type BRUN BOSTON WINDOW.

Warning
When you issue the command BRUN BOSTON WINDOW, any Applesoft program in memory will be lost.

You'll know that you have Boston Window successfully loaded and operational when you see the following message on the display:

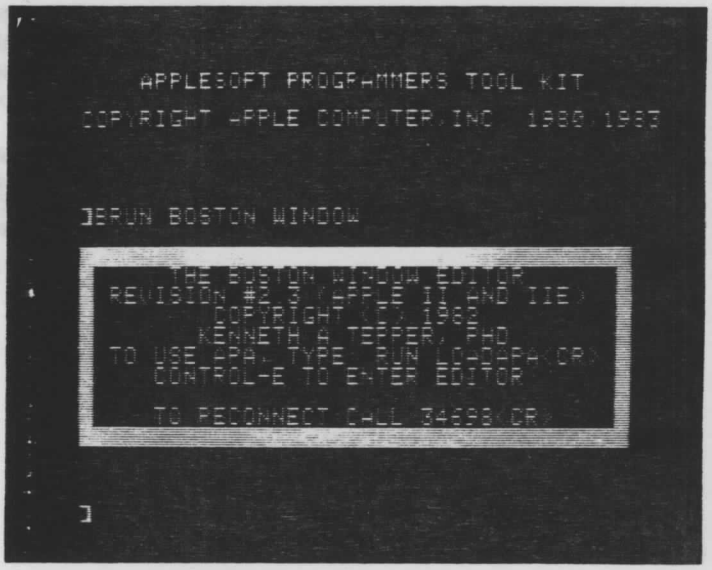


Figure 2-1. Boston Window Opening Message

the [CONTROL] key is labeled [CTRL] on the Apple II and II Plus

[CONTROL]-[E] means Enter Editor

At this point you're not really in the editor; you just have access to it. Get into the editor now by pressing [CONTROL]-[E] (hold down the [CONTROL] key while typing E). Now the screen looks like this:

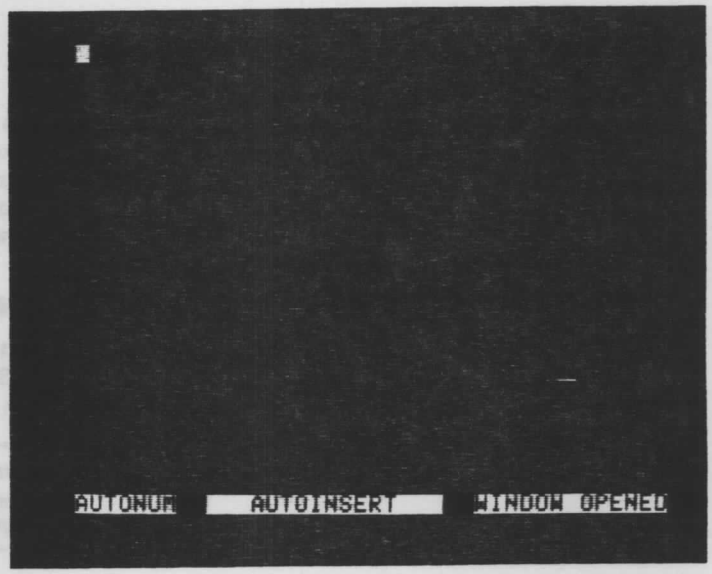


Figure 2-2. Editor Opening Display

At the right edge of the display on the same display line as the cursor you'll see a dot (.). The dot serves notice that you're in the editor. Those three little boxes at the bottom of the display make up the **status line** and provide information about various Boston Window features; you'll learn what each box means as you go through this tutorial.

Getting Rid of the Status Line: If you don't want the status line to appear at the bottom of the display, return to Applesoft by pressing [CONTROL]-[Q]. Then type `POKE 38638, 96`. When you go back to the editor via [CONTROL]-[E], the status line will be gone.

To restore the status line, type `POKE 38638, 152` from Applesoft.

Entering and Changing Single Lines

Boston Window helps you manipulate program lines collectively or individually. This section deals with single lines, and shows you how to enter or change program lines—and parts of lines—in a variety of ways.

Automatic Line Numbering

Boston Window's automatic line numbering feature is ON when you start the program; it assumes you want a line increment of 20. It also assumes that, if the first thing you do after you enter the editor is press RETURN, you want to begin with line number 20. Press RETURN now; the display looks like this:

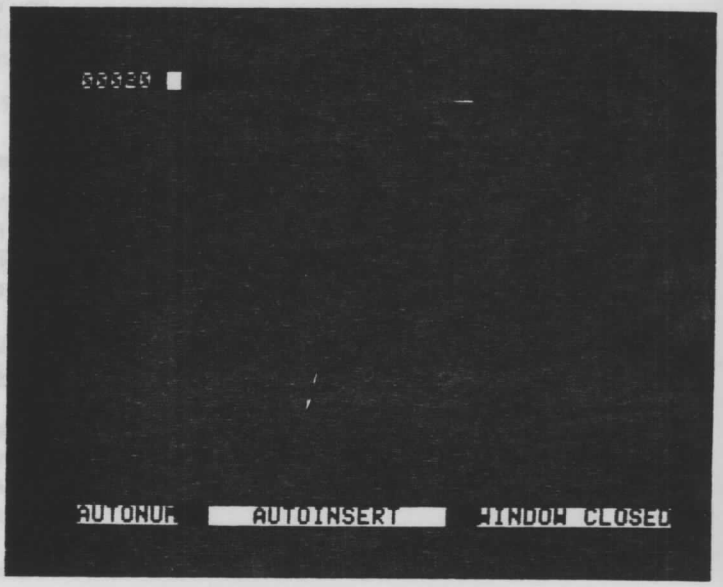


Figure 2-3. Editor Display, Autonumbering ON

The left box on the status line at the bottom of the display lets you know that AUTONUM is on. The rest of this tutorial assumes that the automatic line numbering feature is ON and that its increment value is 20. If you want an automatic increment other than 20, press CONTROL-Q to leave the editor and to return to Applesoft. Then type POKE 38925, <n> where <n> is the increment number.

CONTROL-Q Quits editor

Remember to press CONTROL-E to reenter the editor after you've specified the proper increment.

CONTROL-N means Numbering

If you don't want to use Boston Window's automatic increment feature, just press CONTROL-N while in the editor. This switches the automatic line numbering feature to OFF; the left box on the status line changes its message to MAN NUM. Try it now—but remember to reactivate automatic numbering by pressing CONTROL-N again.

Moving the Cursor Around

Boston Window provides a number of commands for moving the cursor around, and you'll find all of them described here. Of course, in order to move the cursor over characters, you first need some characters over which to have the cursor move. Assuming you've pressed RETURN to make a line number appear, type this into the editor (don't worry about any mistakes you make), but don't press RETURN again:

```
PRINT "COME LET US GO, YOU AND I,"
```

Pure Cursor Moves

You've just typed your first line of code using Boston Window. So far, it looks pretty much like plain old Applesoft. But press CONTROL-B and you'll see the cursor jump to the beginning of the **display line** (not the **program line**).

CONTROL-B means Beginning of line

Jump back to the end of the display line now by pressing CONTROL-E.

CONTROL-E means End of line

Note that these jumps are **pure cursor moves**. That is, these commands move the cursor around without affecting the text at all. Any changes to the text in Boston Window are immediately reflected on the display. What you see is what you get.

As well as making the cursor jump back and forth over whole lines, you can also use Boston Window commands to move the cursor over individual characters. Move the cursor left by pressing CONTROL-A, and then right by pressing CONTROL-S.

CONTROL-A moves cursor left

CONTROL-S moves cursor right

To experience the commands for moving the cursor up and down, you'll need to type in a couple more lines. Press RETURN (the cursor needn't be at the end of the line) and type two more lines of code (again, don't worry about any typos for the moment) so that your display looks like this:

```
00020 PRINT "COME LET US GO, YOU AND I,"
00040 PRINT "WHEN THE EVENING IS SPREAD
 OUT AGAINST THE SKY"
00060 PRINT "LIKE A PATIENT ETHERIZED U
 PON A TABLE;"
```

Note that Boston Window typed the line numbers for you—automatic line numbering is working well! Now press CONTROL-W a few times to move the cursor up the display, and then CONTROL-Z to move the cursor down.

CONTROL-W moves cursor up a line
CONTROL-Z moves cursor down a line

That beeping you hear is Boston Window's way of telling you you're at the top or bottom of a program line—that is, that you've reached the upper or lower limit of the window.

Opening the Window

The term **window** refers to an invisible frame that surrounds each program line. When you begin to type a line of code, the frame is built beginning to the left of and above the line number and extending to the right of and below the last character in the program line. If you could see the frame around line 60's window, it would look like this:

```
00020 PRINT "COME LET US GO, YOU AND I,"
00040 PRINT "WHEN THE EVENING IS SPREAD
 OUT AGAINST THE SKY"
00060 PRINT "LIKE A PATIENT ETHERIZED U
PON A TABLE;"
```

You can add, delete, or change characters within the window (you'll see how in just a moment). But once you start to edit a program line, the window is closed and you're confined in your movements to the area within the window. To open the window press CONTROL-O. Once the window is open, you can move the cursor anywhere in the program.

CONTROL-O Opens the window

To see this more clearly, try this exercise:

1. Open the window by pressing CONTROL-O.
2. Use the various cursor movement keys to put the cursor over the S in SKY in line 40.
3. Now press CONTROL-S to move the cursor until it's over the ending quotation mark.

Look at the right box on the status line at the bottom of the display. The message reads WINDOW OPENED. Press the SPACE bar now—and watch the message change to WINDOW CLOSED. Try moving the cursor up to line 20 or down to line 60 using CONTROL-W and CONTROL-Z. When you reach the window's edge, your computer beeps at you. In fact, the computer beeps any time you try to leave a closed window.

There are a number of ways to open the window so that your cursor can, in hawklike fashion, swoop around the program searching for other text to edit. The simplest way is to press CONTROL-O; you'll discover other methods as you go along. Try it now, and use the cursor movement keys CONTROL-W, -A, -S, and -Z to wander around a bit.

The Cursor Control Diamond

The alert reader (you clever rascal) will have noticed two things by now. First, the keys W, A, S, and Z form a diamond on the keyboard, the four points of which symbolize the direction of the cursor move (see Figure 2-4).

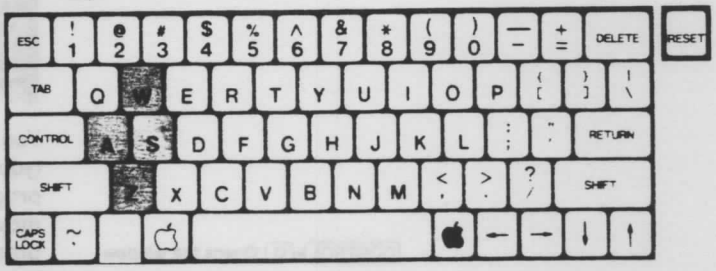


Figure 2-4. W-A-S-Z Cursor Controls

Second, these moves are like the pure cursor moves accessed by the I-J-K-M diamond in escape mode. In fact, escape mode does, indeed, work in Boston Window as long as the window is open (see Figure 2-5).

escape mode cursor moves work in Boston Window

Press (ESC) now and look at the center box on the status line; the message there reads ESCAPE MODE EDIT. Press the (SPACE) bar to leave escape mode, and the ESCAPE MODE EDIT message changes again (and you'll soon learn what the new message means).

Before going on, experiment more with both (W)-(A)-(S)-(Z) and with (I)-(J)-(K)-(M) in escape mode to move the cursor around. They do the same thing, but some people find one method more convenient than another.

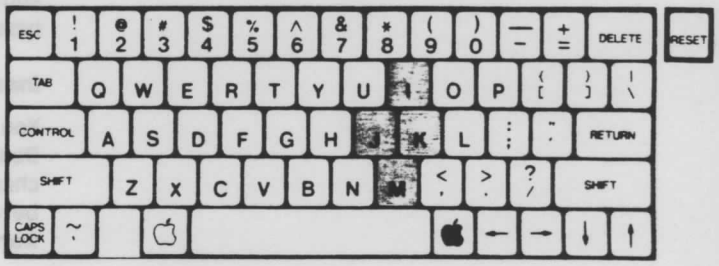


Figure 2-5. (I)-(J)-(K)-(M) Cursor Controls

Inserting Characters

Boston Window provides some easy facilities for adding characters to the middle of a program line. Open the window with (CONTROL)-(O) and use the pure cursor moves keys to move the cursor so that it's over the comma in the first line:

Assuming you aren't in escape mode (and if you are, press the (SPACE) bar to leave it), type a space, followed by the word THEN. Boston Window automatically inserts the characters you type, pushing the cursor and any characters after it to the right. This is on your screen now:

Leaving and Reentering Insert Mode

CONTROL-@ means Overprint or move Over

Boston Window is automatically in insert mode all the time (now you know what the status line's AUTOINSERT message means). If you don't want to be in insert mode, just press CONTROL-@ (@ is SHIFT-P). Any characters you type will replace the next characters, just as would happen in regular Applesoft. Press CONTROL-@ again and you'll restore insert mode. Try it now to see what happens—and notice the change in the message on the status line. Don't worry about all the extra characters you might type; you'll soon learn how to delete them.

Inserting Control Characters

CONTROL-P embeds each control character

You can also embed control characters in your programs using Boston Window. But since Boston Window uses many control characters for commands, it needs to be able to distinguish between command control characters and control characters you want to include in program lines.

Precede each control character you want to embed in the text with a CONTROL-P so Boston Window can know what you want. Embedded control characters show in a listing as highlighted text (black letters on a white or green field). Try it now to get some practice—enter CONTROL-P CONTROL-K, for instance, to embed a CONTROL-K in the text (and, once again, note the change in the status line's middle box). Then read the next section so you can take the control character out again!

Deleting and Restoring Characters

Ordinarily, pressing the ← and → keys in the Apple II makes the cursor pass over characters without changing what appears on the display. But press the ← key in Boston Window and characters start to disappear. At the same time, all the characters from the cursor to the end of the program line move to the left to fill the empty space.

← deletes, → restores characters

Press ← now three or four times to see this happen. If you practiced typing in control characters with CONTROL-P, use this deleting feature to remove them. Then press the → key a few times, and the characters reappear (they were never lost—just stored in a **buffer**). You can delete and restore up to 255 characters using the arrow keys.

To move the cursor to the left or right without deleting or restoring characters, use J and K in escape mode or CONTROL-A and CONTROL-S.

It turns out that the text of line 20 is incorrect (it's supposed to be the first line from T. S. Eliot's poem "The Love Song of J. Alfred Prufrock"). Move the cursor to the space between the words COME and LET. Then use ← to erase the word COME.

Deleting Characters Under the Cursor

CONTROL-D means Delete character under cursor

Just as ← deletes characters to the left of the cursor, CONTROL-D deletes the character under the cursor. Luckily there is still a minor problem with line 20, just waiting to be corrected via CONTROL-D: there shouldn't be a space between the opening quotation mark and the word LET. Move the cursor over that space and press CONTROL-D; the space disappears into the delete buffer.

If you envision the cursor as lying to the left of the character over which it appears (sort of sandwiched between two characters), then you can think of ← as deleting the character to the left of the cursor and CONTROL-D as removing the character to the right of the cursor. Play with CONTROL-D a bit and you'll see how it differs from ←'s action.

A Brief Note About Retrieving Characters: Boston Window has several storage areas called buffers where characters are temporarily placed for safekeeping. When you delete characters from a program, they are moved into the delete buffer so that you can later retrieve them via the → key.

Retrieving characters deleted via CONTROL-D, however, can have some strange results; they'll come out backwards. For instance, if you delete the word PRINT with CONTROL-D's and then use →, you'll get TNIRP!

Deleting to End of Line

CONTROL-Y deletes to end of line

Boston Window offers a handy command for deleting text from the cursor to the end of the program line—CONTROL-Y. This command can save you lots of time when you want to remove REM statements selectively, or for those situations where you've modified a line of code and have a bunch of characters left over. Characters deleted via CONTROL-Y go into the buffer, so you can get them back if you need to.

Note to Mass REM Deleters: If you want to strip your program of all REM statements, use Applesoft Programmer's Assistant's &C command. See Chapter 1 for the details.

Deleting Whole Lines

CONTROL-L means delete Line

There's one more command you can use to delete text into a buffer; press CONTROL-L and the entire program line disappears into the delete buffer. This feature is useful if the line you've typed is so messed up that it's beyond repair. Once again, as with the other deletion commands, the line isn't really gone; it's in the buffer. You can press → and hold down the REPT key until the line is back on the display.

Completing a Program Line

RETURN finishes work on a line

No matter where the cursor is in a program line, when you press RETURN the whole line goes into Applesoft's program memory. Move the cursor to the middle of line 60 now and press RETURN—none of the characters will be lost.

Actually, when you press RETURN a number of things happen. The current program line goes into Applesoft's program memory, a new line number with the appropriate increment is written to the display, and the cursor is placed so that you can begin to type the next line, with a window closed around the new program line.

Inserting New Program Lines Between the Old

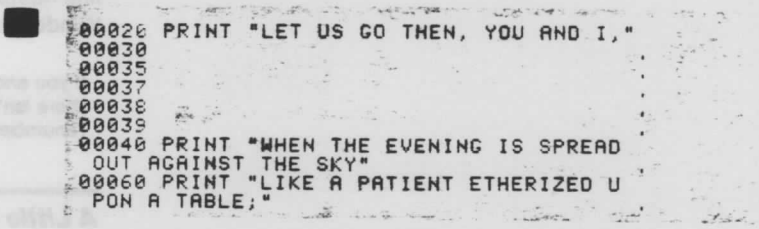
You've already seen how Boston Window's autonumbering works to type new consecutive program lines. You can also use it to type new program lines between existing ones.

Move the cursor so that it's somewhere in line 20. Then press RETURN. If autonumbering is in effect (as it should be, unless you shut it off by pressing a CONTROL-N), Boston Window types line number 30 for you and correctly places the cursor so you can type new code.

When you press RETURN, Boston Window takes the line number that it is currently on (line 20) and attempts to set up a new line number for you. Ordinarily, the new line number would be the old line number plus 20, which is the standard increment. But line number 40 is already being used; so Boston Window takes the current line number (20) and averages it with the number of the next existing line (line 40). Some quick arithmetic:

```
20 + 40 = 60
60 / 2 = 30
```

Boston Window assigns the number 30 to the next line. If you press RETURN again, Boston Window leaves line 30 blank and averages 30 with 40—and sets you up to type line number 35. Press RETURN a few more times; when Boston Window runs out of integers to use for line numbers, it just beeps at you.



If autonumbering is not in effect (that is, if you've disabled it by pressing CONTROL-N), then Boston Window just opens a space between the current and next program line and waits for you to type in your own number.

Blank Lines Are Harmless

All those blank lines on the display are harmless. You can ignore them, make them go away by pressing a few CONTROL-L's (CONTROL-L deletes a program line), or get rid of them by leaving and restarting Boston Window with CONTROL-Q CONTROL-E. Try both of these methods now, just for the practice.

Adding a Lower Line Number

CONTROL-^ sets up immediately preceding line

Boston Window lets you add line numbers before existing ones as well as after them via the CONTROL-^ command. To see what this is all about, move the cursor so that it's at the start of line 20. You've just seen that by pressing RETURN you can type a new program line between lines 20 and 40; to type a program line *before* line 20, press CONTROL-^. Boston Window makes a space *above* line 20 (by scrolling all text down one line) so you can begin typing more code.

Notice that Boston Window doesn't assign a line number here; if you want one, just press RETURN—and you'll be typing code at line 10 (the average of 20 plus 0, since 0 is the lowest line number you can use in Applesoft).

Ordinarily Boston Window sets up a line number for you when you use CONTROL-^. It didn't give a line number here because you typed a new lowest line; to Boston Window, a new lowest line is a special situation. To get a line number, either type in your own or just press RETURN.

You can use CONTROL-^ anywhere in the program. If there is no room for a new line number preceding the current one (for instance, you already have a line number 99 and you're currently in line number 100), pressing CONTROL-^ just makes Boston Window beep at you.

If you encounter a situation where you need a new line number and there isn't room for one, use the Applesoft Programmer's Assistant's renumbering feature described in Chapter 1.

A Little Practice

Leave the editor now (CONTROL-Q) and list the program so you can see your handiwork. Then run it. if you don't like how the display looks, go back into the editor by pressing CONTROL-E and fix the code. When you're through, go on to the next section to learn about the rest of Boston Window's features.

Dealing with Larger Programs

To experience the rest of Boston Window's features, you'll need to have a longer program in memory. Load (but don't run) the MAXWELL program. Then enter Boston Window (CONTROL-E).

Scrolling Through Your Program

With the window open, the cursor can zip anywhere in the program. When you reach the bottom of the display, the code scrolls up one line at a time. Try using CONTROL-Z at the display's bottom to see this happen.

Having to hold down keys just to list through a program is somewhat tedious; to have Boston Window do it for you, press CONTROL-V. CONTROL-V moves the cursor to the bottom of the display and scrolls the program continuously until the listing's end, much like Applesoft's LIST command. To stop the listing at any point, press any character key. To control the listing, experienced users of Boston Window keep the CONTROL key pressed with one finger, using another finger to press the V.

CONTROL-V scrolls down through program; pressing any character key stops it

To Use or Not to Use CONTROL-C: When you are listing an Applesoft program, you would ordinarily use CONTROL-S to pause and continue the listing; if you wanted to terminate the listing, you would use CONTROL-C. Such is not the case with Boston Window's CONTROL-V scrolling command. When you stop the scrolling by pressing a character key, the scrolling is in fact terminated. There is no need for a CONTROL-C; in fact, Boston Window uses CONTROL-C to start a special Copy Characters sequence (see below). If you accidentally press CONTROL-C and strange things start appearing on your display, press CONTROL-C again followed by CONTROL-O, and everything will be OK.

CONTROL-T scrolls towards the Top; pressing any character key stops it

Unlike plain vanilla Applesoft, Boston Window also allows you to scroll through a program backwards. Press CONTROL-T and Boston Window places the cursor at the top of the display and begins scrolling until it reaches the start of the code. As with CONTROL-V, any character key stops the scrolling and another press of CONTROL-T starts it again.

In Case You Wondered About Highlighted Characters: In scrolling through MAXWELL, you will have noticed all those highlighted characters. Each one is an embedded control character. It happens that most of them are special commands used by the High Resolution Character Generator, described in detail in Chapter 6.

Skipping Through Your Program

Boston Window provides a quick way of moving from one section of a program to another: it's called the GOTO command.

CONTROL-G lets you go anywhere in the program

Assume that you want to edit line number 2000. Press CONTROL-G and Boston Window responds by displaying `GOTO:` in highlighted text near the top left of the display. Now type the number `2000` and press RETURN. Boston Window obediently shows a displayful of code beginning with line 1990 (the line of code immediately preceding the line specified), with the cursor positioned over the first character of line 2000.

Of course, now that you're at line 2000 you don't have to edit it; you can use GOTO just to look at different sections of code.

If the line number you specify with CONTROL-G doesn't exist, then the cursor goes to the next higher numbered line. If the specified line is higher than the final line number in the program, then the cursor goes to the program's final line. If the line number you specify is lower than the lowest line, then the cursor goes to the first line of the program.

A Quick Zip Tip: To get to the first line of the program, type

CONTROL-G RETURN

To get to the last line, type

CONTROL-G 63999 RETURN

Finding a String

CONTROL-F Finds a string

You can also have Boston Window find a particular string of characters located anywhere in the program. Press CONTROL-F followed by any string of characters (up to 34 characters long), and press RETURN. Boston Window moves the cursor to the first occurrence of the string beyond the cursor's present position.

Assuming the cursor is at the start of line 10 in the current program (and if it isn't, put it there by typing CONTROL-G 10), press CONTROL-F. FIND: appears in highlighted text just below line 10. Now type the word WAVE and press RETURN. The cursor moves down the display to line 200, just beyond the word WAVE.

If you press CONTROL-F again, Boston Window tries to find the next occurrence of the word WAVE following the cursor's current position (FIND always searches ahead). It turns out that the word doesn't appear again in the program; so after trying its best to find it, Boston Window signals by beeping that it can't find a new WAVE.

FIND does a literal search for its strings. That means that if you tell Boston Window to FIND the letters IN, it will stop at the IN in IND., INITIALIZE, and so on. If you want to find *only* the whole word IN, then you must specify the spaces separating the word from its neighbors. Further, if you specify IN (uppercase characters), then FIND won't discover lowercase in. To the FIND command, uppercase is uppercase and lowercase is lowercase. You must search for each separately.

To Search for Another String

If you've found the string you're searching for and you want to give FIND another string to search out, press CONTROL-G RETURN. This puts the cursor at the start of the first line, with FIND ready for new material. If you press CONTROL-F now and just press RETURN, the old material in the buffer is retained and Boston Window looks for the most recently specified string.

Searching for a Control Character: When using CONTROL-F to find a control character, you must press CONTROL-P before typing the control character for the search.

A Wildcard Character

CONTROL-W means Wildcard

Typing CONTROL-W as a character in the FIND string enters a highlighted blank space in the string. The highlighted blank space is a wildcard character, matching any character it finds in that position in the string. You can have as many wildcard characters in a string as you want. Try this:

What You Type	What Happens
1. Press CONTROL-G	
2. Press RETURN	Moves cursor to top of program
3. Press CONTROL-F	
4. Type WA and press CONTROL-W CONTROL-W	Tells Boston Window to look for any string at least four characters long starting with characters WA
5. Press RETURN	Begins the search; stops at end of WALK in line 100
6. Press CONTROL-F	Continues search; stops at end of WAVE in line 200

CONTROL-R means Replace

Replacing a String

Using CONTROL-R, a variation of the FIND command, you can replace some or all of the occurrences of a specified string with another string.

Replacing Selectively

In the following example you'll change the word GOSUB to GOTO (to Boston Window, a command word is just another string). Here's what to do:

What You Type	What Happens
1. Press CONTROL-G	
2. Press RETURN	Moves cursor to top of program
3. Press CONTROL-F	
4. Type GOSUB	Tells Boston Window to look for the string GOSUB
5. Press CONTROL-R and type GOTO	Tells BW to replace GOSUB with GOTO
6. Press RETURN	Cursor moves to space following first occurrence of the word GOSUB (line 40)

Now you have a choice. You can press another CONTROL-F, leaving the GOSUB in line 40 intact and causing the cursor to move to line 80's GOSUB. Instead, press CONTROL-R, and *voila!* the GOSUB becomes a GOTO.

You can continue to press CONTROL-F (to find the string again) or CONTROL-F CONTROL-R (to both find and replace the string) until you go through all occurrences of GOSUB in the program.

Replacing Globally

Boston Window offers you a final variation on FIND: the global replace option. It lets you replace all occurrences of a given string (or any group of characters) in a program with a single command. Try this:

1. Press [CONTROL]-[G]
2. Press [RETURN] (gets you to start of program)
3. Press [CONTROL]-[F]
4. Type REM
5. Press [CONTROL]-[R]
6. Type RAM (but don't press [RETURN] yet)

[CONTROL]-[G] replaces Globally; pressing any character key stops it

You're going to replace all occurrences of REM with RAM—but instead of pressing [RETURN], press [CONTROL]-[G]. In this context, [CONTROL]-[G] stands for Global Replace. The cursor zips through the program, changing before your very eyes all REMs to RAMs.

If you change your mind you can stop this wholesale replacing while it's still going on by pressing any character key; Boston Window signals that it understands your cease-and-desist command by beeping at you. But any changes that Global Replace made up to the time you stopped the process remain in effect.

To replace all occurrences of a given string with the null string—that is, a string with no characters—follow the [CONTROL]-[R] command directly with [CONTROL]-[G].

Copying Code or Strings

Quite often in programming you come across situations where you need to use the same code or strings again and again, but for one reason or another you don't want to use subroutines. To reduce the drudgery of repetitive typing, Boston Window offers the copy buffer.

[CONTROL]-[C] means Copy or Capture

Use [CONTROL]-[G] (which means GOTO in this context) to move the cursor to line 910. Now press [CONTROL]-[C] and use [→] (actually any character key will do) to move the cursor until it's over the semicolon. As the cursor moves, the characters over which it passes change to highlighted text to let you know that they have been copied into the copy buffer (look at the status line; it, too, tells you when COPY is operating). When the cursor is over the semicolon, press [CONTROL]-[C] again; the copied characters now appear in normal video, and the copy buffer is closed.

If you go clear to the end of the program line, the copy buffer closes itself. The buffer holds up to 255 characters.

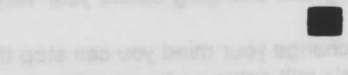
CONTROL-I means Insert

Now move the cursor to line 1100 and press RETURN to set up a new line. Then press CONTROL-I and the contents of the copy buffer (line 910's code without the semicolon) are inserted into line 1105.

The buffer isn't empty; it still holds the characters you just put into it. To prove it, do this now:

1. Move the cursor to line 60.
2. Use CONTROL-E to move to the end of the line.
3. Add a colon (:).
4. Press CONTROL-I again.

The line now looks like this:



You can use the contents of the copy buffer again and again; it won't change until you issue another CONTROL-C command.

Half-Screen Mode for Debugging

CALL 34689 gets you half-screen mode

Boston Window's final feature is half-screen mode. By issuing the Applesoft command CALL 34689 you can make debugging easier by watching the program run in the bottom half of the display while the code is visible in the top half.

Turning Off the Status Line: You can't turn off the status line while the program is in half-screen. Return to full-screen by typing CALL 34698 from Appiesoft (CONTROL-Q puts you back in Applesoft). Then type POKE 38638, 96 to turn off the status line.

Half-screen functions a little differently from full-screen. To see it work, first type NEW from Applesoft. Then, using autonumbering and the copy buffer to make the job quick and easy (see the box below if you're having trouble), type the following short program while you're still in full-screen mode:

```
00020 FOR X=1 TO 1000
00040 PRINT X;
00060 NEXT X
00080 GOTO 20
00100 REM
00120 REM
00140 REM
00160 REM
00180 REM
00200 REM
00220 REM
00240 REM
00260 REM
00280 REM
00300 REM
```

About That Shortcut: In case you're having trouble figuring out how to use the copy buffer to type in the sample program, here's how to do it.

1. Type the first four lines as you normally would, using the automatic numbering feature to type the line numbers for you. If this feature isn't working, press (CONTROL)-(N) to turn it on.
2. When you get to line 100, type REM but don't press (RETURN).
3. Move the cursor to the R in REM.
4. Press (CONTROL)-(C) to start putting characters into the copy buffer.
5. Press the (SPACE) bar (or any character key) to copy the word REM.
6. Press (CONTROL)-(C) again.
7. Press (RETURN) and then (CONTROL)-(I).

As you can see, the last step in this series made line number 120 appear, with REM typed after it. Repeat step 7 a few more times and you're done.

Now leave the editor via (CONTROL)-(Q) and issue a CALL 34689 to set half-screen mode. As soon as you type this command and press (RETURN), the display flashes and only lines 20 through 240 are visible; that's all that will fit on the top half of the display.

Now run the program. The bottom half of the display quickly fills up with numbers. When you're ready, press CONTROL-C to stop the action; then get into the editor via CONTROL-E.

Switching between CONTROL-V (scroll down) and CONTROL-T (scroll up) will show you that all the code is still there; it's just somewhat hidden.

CALL 34698 restores full-screen

To restore full-screen again, type CALL 34698 from Applesoft (use CONTROL-Q to leave the editor and get back to Applesoft).

▲ **Warning**

Don't use Applesoft's TEXT command to reset the display window from half-screen to full-screen. TEXT does funny things to Boston Window, making it in turn do strange things to the system memory, and ultimately causing everything to go someplace indeterminate where it might never, never be found.

To recover from TEXT, type CALL 34689 to restore the half-screen display; or type CALL 34698 to set up full-screen display.

Section 2

Boston Window Reference

This section contains information you'll want to refer to from time to time as you use Boston Window. It covers the material discussed in the tutorial section but in a more condensed form; you'll find this part of the chapter especially useful as an expansion of the notes from the Quick Reference Card at the back of the manual.

Additionally, there are comments here on the general structure of Boston Window, hints for troubleshooting unusual problems that might pop up, and special options for more advanced programmers.

Boston Window and Line Numbers

When Boston Window begins editing a line of code, the line is first deleted from Applesoft's program memory. When editing is completed the program line is inserted into program memory. Therefore, it's extremely important that you type program line numbers correctly. Most people type line numbers in the manner about to be described anyway; these paragraphs are included just to be on the safe side.

The editor assumes that the line number begins at the display's left margin. The editor also assumes that the line number is contained in the first five columns of the display line. It reads up to five characters or until it reads a nonnumeric character, whichever comes first.

In order for a line number to be valid it must follow the same rules that apply to line numbers in Applesoft BASIC, with three additional constraints:

- The line number must start in the left margin (column 1).
- The line number cannot have any spaces in it.
- The line number must be contained in the first five columns.

Warning

Never change a line number to an existing line number. If you do, the original line of code is destroyed and replaced by the new one. Further, if you attempt to swap the positions of two or more program lines by changing the line numbers on the display, you put yourself in danger of losing *all* the lines you're trying to reposition.

Troubleshooting Boston Window

Sometimes Boston Window might act in ways that surprise you. In the vast majority of cases, you can trace the strange behavior to some command you have inadvertently given or to some function you haven't used enough to understand fully. Here are some such confusing situations you might come across, with suggestions for clearing the problems up.

If You Want to Initialize a Disk...

Be sure the SYSTEM MASTER version of DOS is in memory. Initializing a blank disk while Boston Window is hidden in high memory—between DOS itself and the file buffers—will result in a nonstandard copy of DOS being written on the disk. As a result, there is a reduction of available memory to your application program.

If You Try Using Boston Window with Integer BASIC...

You can execute Boston Window from Integer BASIC, but the program won't work on Integer code. Once you've got Boston Window in memory, however, you can load an Applesoft program and Boston Window will work fine.

If Nonsense Characters Fill the Screen ...

If you get a screen full of **garbage** when you enter the editor, you probably have somehow gotten yourself into the Monitor or Integer BASIC. If this happens, press CONTROL-Q to leave the editor and type FP to get into Applesoft. If you type FP and nothing happens, you'll have to restart the system. Life is filled with such minor disappointments; hang in there.

If Boston Window Seems to Disappear...

Boston Window occasionally loses its connection to the rest of the system (for instance, when you press CONTROL-RESET) and seems to vanish into The Big Buffer In The Sky. If this happens, type `CALL 34698` from Applesoft and Boston Window is usually accessible again.

If Random Program Lines Appear...

Random lines at the display's bottom means that Boston Window has been partially disconnected from the system, probably because you issued a PR# command. Just type `CALL 34698` from Applesoft to reconnect.

If You Use Global Replace...

Replacing globally can be a dangerous practice. Be sure that you really want to replace *every* occurrence of a given string before using this feature—and even then, stand ready to halt the process by pressing the SPACE bar (or any other character key). Also remember that, if you change a word like *in* to *out*, you might end up with strange *out*formation in your file! Be sure to indicate whole words by including spaces before and after the characters you want changed.

If You Use Half-Screen Mode...

Don't use Applesoft's TEXT command to reset the screen window from half-screen to full-screen. To recover from TEXT, type `CALL 34689` to restore the half-screen display; or type `CALL 34698` to set things up for full-screen editing.

If Your Program Has Lines Longer than 239 Characters...

There are special Applesoft utility programs available that allow you to type programs with lines longer than 239 characters. If the program you are editing with Boston Window has been written using such a utility, and if you attempt to edit a line containing more than the normally allowed number of characters (239 including embedded spaces), the line will be destroyed.

Before editing such a line, break the line into shorter segments from Applesoft rather than from Boston Window. Then you can safely enter the editor and change away. For other ways of handling long lines, see the next section, on **packed lines**.

Special Packed Line Options

Some programmers like to save memory space by combining several statements on the same program line. If you are one of those programmers, then you know the frustration of trying to change an element in a long program line. Quite often you'll be copying the line into memory, having made your changes, when Applesoft begins beeping at you—letting you know your line now has more than the 239-character limit. Boston Window gives you three ways to beat this numbers game, in the form of three different POKE commands.

Reducing "PRINT" to "?"

Boston Window's first byte-saver changes all PRINT statements to the familiar question mark (?), saving four characters for each instance of use. The command, issued from Applesoft, is `POKE 37781, 186`. To restore the normal display, type `POKE 37781, 0`.

`POKE 37781, 186` displays `PRINT` as `?`

Deleting All Extra Spaces

POKE 37048, 160 eliminates nonquoted spaces

The second POKE command in this series does away with embedded spaces in a program line. Normally, the editor sends program lines to Applesoft as they appear on the display; thus any embedded spaces count against the 239-character total. But with `POKE 37048, 160` you instruct the editor to delete all spaces not in a quoted string as a line is sent to Applesoft. In a line of, say, 50 words, that's a savings of 49 characters! To restore normal operations, use `POKE 37048, 0`.

Note that Boston Window takes out the extra spaces only as it sends a line to Applesoft and not while the editor is displaying the line—you won't see any changes on the display. A word of warning here, though: unless you enclose the text in quotation marks, the spaces between words in REM statements are also eliminated, resulting in virtually unreadable comments. The solution is either to put all the text of REM statements in quotation marks, or to use the third special POKE command.

Making the Display Compact

POKE 37789, 186 compacts the display

The final compacting method leaves spaces in REM statements and quoted strings, but takes out spaces between command words in the display. You'd use this method, which you access with `POKE 37789, 186` and cancel with `POKE 37789, 0`, when your program line takes up more than seven display lines.

Dense program lines require more screen lines to display than do normal program lines, and this length is amplified by the editor's display formatter, which adds extra spaces for clarity. In order to accommodate the size of the buffers the editor uses for scrolling, the editor sets a limit of seven on the number of screen lines a single program line can occupy.

If a program line scrolls onto the screen and is longer than seven screen lines, the editor truncates the screen display; the copy of the line in program storage is unaffected. Issuing `POKE 37789, 186` changes the way the editor formats the display, so that even the longest program line fits in seven lines.

Using this method, however, jams all the command words next to each other in the display, making program lines very difficult to read. Unless you are writing *very* long lines and are concerned with seeing the entire line on the display, it makes more sense to use the first two POKE options—and remember to put your REM text in quotation marks.

Quote those comments!

Command Summaries

Presented below are lists of Boston Window's commands, categorized in a number of different ways. First, the commands are grouped as to functions—cursor movement, deletion, and so on. Later is an alphabetical list of all control character commands. Finally you'll find a diagram showing Boston Window's many cursor movement commands. You'll also find these summaries on the handy tear-out reference card at the back of this manual.

Functional Command Summary

Editor Access

Enter editor	CONTROL-E
Quit editor	CONTROL-Q

Cursor Movement

Open window	CONTROL-O
Move left one character	ESC J or CONTROL-A
Move right one character	ESC K or CONTROL-S
Move up one line	ESC I or CONTROL-W or -K
Move down one line	ESC M or CONTROL-Z or -J
Move to left edge of screen	CONTROL-B
Move to right edge of screen	CONTROL-E
Scroll toward top of program	CONTROL-T
Scroll toward bottom of program	CONTROL-V

Delete

Delete character left	CONTROL-H or ←
Delete character at cursor	CONTROL-D
Delete program line	CONTROL-L
Delete to end of program line	CONTROL-Y

Restore

Restore last character deleted	CONTROL-U or →

Copy

Open, close copy buffer	CONTROL-C
Insert copy buffer contents	CONTROL-I

Find, Replace

Find specified string	CONTROL-F
Add wildcard character	CONTROL-W
Replace string with specified string	CONTROL-R
Global Replace (after CONTROL-F CONTROL-R)	CONTROL-G

Start New Line

Start new higher line	CONTROL-M or RETURN
Start new lower line	CONTROL-^

Miscellaneous Functions

Autonumbering ON-OFF	CONTROL-N
Autoinsert ON-OFF	CONTROL-@
GOTO specified line number	CONTROL-G
Insert control character	CONTROL-P

Alphabetic Command Summary

CONTROL-A	Moves cursor left one character—same as ESC J
CONTROL-B	Moves cursor to left edge of current display line
CONTROL-C	Opens/closes copy buffer
CONTROL-D	Deletes character at cursor position, moves all text left one space
CONTROL-E	From Applesoft, enters editor; from editor, moves cursor to right edge of current screen line
CONTROL-F	FINDs specified string
CONTROL-G	GOTO specified line number; at end of CONTROL-F CONTROL-R sequence, does Global Replace
CONTROL-H	Deletes character to left of cursor, moves all text left one space—same as ←
CONTROL-I	Inserts contents of copy buffer at cursor position
CONTROL-J	Moves cursor down one line—same as CONTROL-Z, ESC M
CONTROL-K	Moves cursor up one line—same as CONTROL-W, ESC I
CONTROL-L	Deletes program line containing cursor
CONTROL-M	Opens window on current program line, closes window on new line; if autonumbering is in effect, assigns line number to new line higher than current line—same as RETURN key press

CONTROL-N	If autonumbering is ON, turns OFF; if autonumbering is OFF, turns ON
CONTROL-O	Opens window
CONTROL-P	Use before adding control character to text; control character displayed in inverse video
CONTROL-Q	Leaves the editor and returns to Applesoft with cursor at bottom left of screen
CONTROL-R	Used after CONTROL-F to establish replacement string or to effect the replacement
CONTROL-S	Moves cursor right one character—same as ESC K
CONTROL-T	Moves cursor toward top of window or scrolls to start of program; stopped by press of any character key
CONTROL-U	Restores one character from delete buffer—same as →
CONTROL-V	Moves cursor toward bottom of window or scrolls to end of program; stopped by press of any character key
CONTROL-W	Moves cursor up one line—same as CONTROL-K, ESC I; wildcard character in FIND string
CONTROL-X	Cancels current command (FIND or GOTO)
CONTROL-Y	Deletes into buffer all characters from cursor to end of program line
CONTROL-Z	Moves cursor down one line—same as CONTROL-J, ESC M
CONTROL-@	If autoinsert is ON, turns OFF; if autoinsert is OFF, turns ON
CONTROL-^	Opens window on current line; closes window on line to have lower line number than current line; if autonumbering is in effect, assigns line number to new line lower than current line

Special CALLs and POKEs

Reconnect Boston Window, full screen	CALL 34698
Reconnect Boston Window, half screen	CALL 34689
Set AUTONUM to stipulated increment	POKE 38925, <n>
Delete nonquoted spaces (buffer)	POKE 37048, 160
Restore to normal	POKE 37048, 0
Delete nonquoted spaces (screen)	POKE 37789, 186
Restore to normal	POKE 37789, 0
Display PRINT as ?	POKE 37781, 186
Restore to normal	POKE 37781, 0
Turn off status line	POKE 38638, 96
Display status line	POKE 38638, 152

Functions of Special Keys

ESC	Turns on escape mode
RETURN	Opens window on curent line, inserts blank line below current line, and closes editing window around blank line
←	Removes character left and puts into delete buffer
→	Restores last character put into delete buffer

Cursor Moves Diagram

Here are Boston Window's pure cursor move keys. Pressing ESC turns on the familiar escape mode; see the *Applesoft BASIC Programmer's Reference Manual* for details.

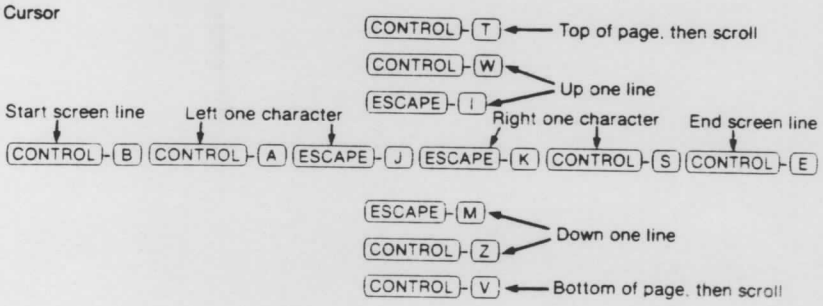


Figure 2-6. Boston Window Cursor Moves Diagram

Page 0 Memory Usage

The following table shows the hexadecimal (HEX) and decimal (DEC) memory locations on page 0 used by Boston Window. Where two bytes are named, low byte is listed first.

HEX	DEC	PURPOSE
E3	227	Horizontal cursor position
EB	235	Vertical cursor position
EC-ED	236-237	Cursor screen address
EE-EF	238-239	Line number holding cursor
F9-FA	249-250	Address of last keyword read
FB-FC	251-252	Screen address for scrolling
FD-FE	253-254	Address of program line holding cursor

Boston Window is positioned in memory just below DOS, with DOS's input and output buffers shifted down to accommodate it. Boston Window occupies about 5 kilobytes from $8781 (decimal 34689) to $9CFF (decimal 40191); it sets the highest address for DOS buffers at $8780 (decimal 34688).

Boston Window for the Apple IIe

Boston Window for the Apple IIe

The Boston Window is a powerful Applesoft BASIC program editor. It's like a word processor for programs, allowing you to type and modify Applesoft programs quickly and easily.

Using this editor you can, among other things, scroll both backward and forward through a program, copy all or part of a program line, insert or delete characters within a line, and search for and change statements and text strings either once or throughout your program. Coupled with Applesoft Programmer's Assistant (see Chapter 1), Boston Window creates a wonderfully simple and complete environment in which to program.

To Use Boston Window and APA Together: To use the features of both programs at the same time, you need to run the Boston Window Program before you start up APA. To do that, first type BRUN BOSTON WINDOW and press RETURN. Then type RUN LOADAPA and press RETURN.

Boston Window is designed for people with some Applesoft programming experience; this chapter assumes that you have been programming in Applesoft for a while. As with any complex program with a lot of features, there's a learning curve with Boston Window—it will take you a while to learn to use everything it has to offer. But it will be worth it!

This Chapter for IIe Users Only: This chapter is geared specifically to users of the Apple IIe computer. If you are using an Apple II, turn to the chapter labeled (as you might reasonably expect) Chapter 2.

How This Chapter Is Organized

This chapter has two sections. Section 1 is a tutorial and should be read while you're sitting at your computer; if you follow it straight through to the end, you'll be walked through most of Boston Window's commands and options. If you're a first-time user of the program, we recommend you go through this section at least once to get the hang of using Boston Window.

Section 2 is a reference section; it contains summary tables of all commands and options, a chart of the cursor movement keys, and miscellaneous technical data.

Where The Name Comes From: The *Boston* part of the program's name comes, naturally enough, from the town where its author—Kenneth A. Tepper, Ph.D.—lives (an obscure suburb of New York, some 218 miles to its northeast; the town is rumored to have had some significance during the Revolutionary War). *Window* refers to an invisible frame that surrounds each program line while you're in the process of editing it. There'll be more about the window concept as we go along.

Section 1

Boston Window Tutorial

This tutorial assumes you are already familiar with Applesoft; it makes no attempt to teach the basics of programming. If you have no experience with Applesoft, spend some time with the *Applesoft Tutorial* manual and get the elementary stuff out of the way. Then come back here; we'll wait for you.

Boston Window works both with and without the Apple IIe 80-Column Text Card in ACTIVE-80 mode. Switching back and forth disconnects Boston Window, but it's a simple matter to reconnect: just type `CALL 34698`. For demonstration purposes, this tutorial assumes that you are using the Apple's native 40-column format.

Starting Boston Window

Getting Boston Window started is really simple. Just insert the DOS Programmer's Tool Kit Volume I disk into the appropriate disk drive and type `BRUN BOSTON WINDOW`.

▲ **Warning**

When you issue the command `BRUN BOSTON WINDOW`, any Applesoft program in memory will be lost.

You'll know that you have Boston Window successfully loaded and operational when you see the following message on the display:

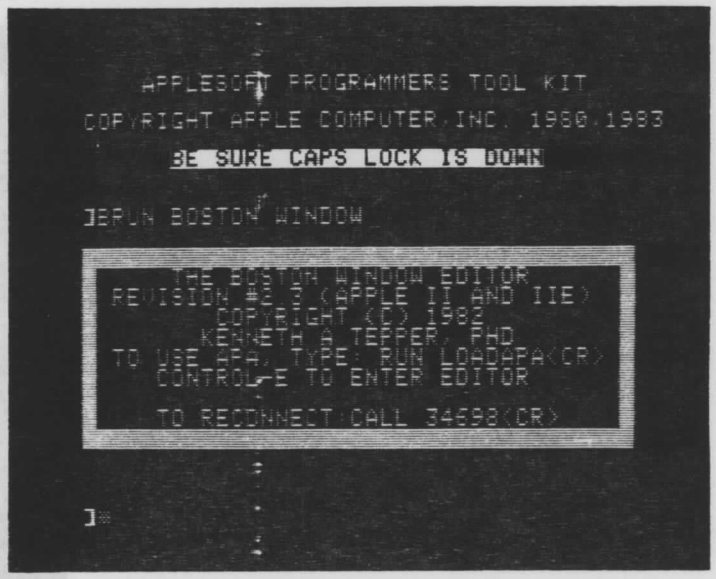


Figure 2e-1. Boston Window Opening Message

CONTROL-E means Enter Editor

At this point you're not really in the editor; you just have access to it. Get into the editor now by pressing CONTROL-E (hold do the CONTROL key while typing E). Now the screen looks like this.

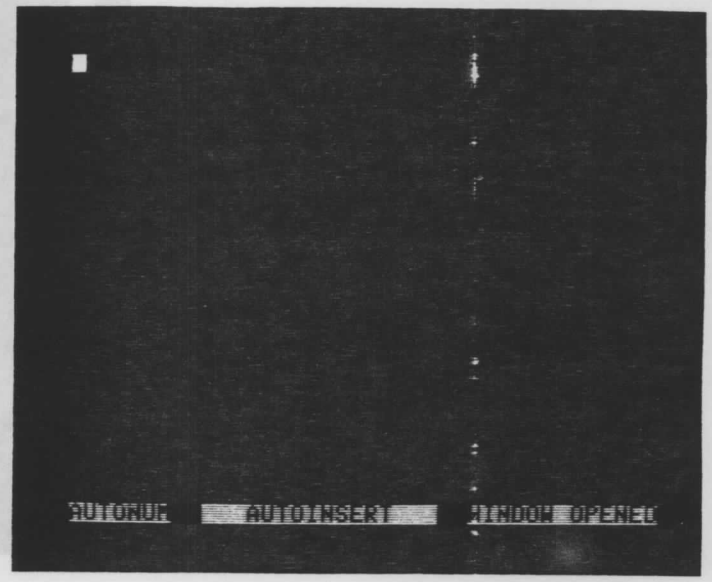


Figure 2e-2. Editor Opening Display

At the right edge of the display on the same display line as the cursor you'll see a dot (.). The dot serves notice that you're in the editor. Those three little boxes at the bottom of the display make up the **status line** and provide information about various Boston Window features; you'll learn what each box means as you go through this tutorial.

Getting Rid of the Status Line: If you don't want the status line to appear at the bottom of the display, return to Applesoft by pressing CONTROL-Q. Then type `POKE 38638, 96`. When you go back to the editor via CONTROL-E, the status line will be gone.

To restore the status line, type `POKE 38638, 152` from Applesoft.

Entering and Changing Single Lines

Boston Window helps you manipulate program lines collectively or individually. This section deals with single lines, and shows you how to enter or change program lines—and parts of lines—in a variety of ways.

Automatic Line Numbering

Boston Window's automatic line numbering feature is ON when you start the program; it assumes you want a line increment of 20. It also assumes that, if the first thing you do after you enter the editor is press (RETURN), you want to begin with line number 20. Press (RETURN) now; the display looks like this:

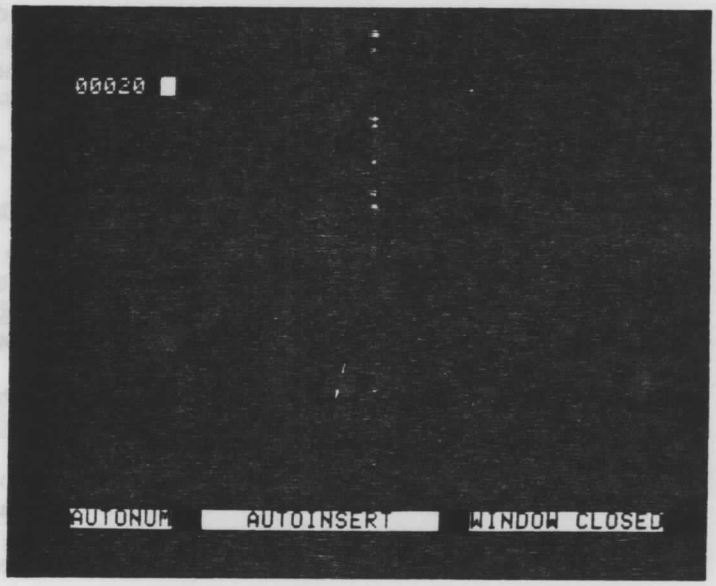


Figure 2e-3. Editor Display, Autonumbering ON

The left box on the status line at the bottom of the display lets you know that AUTONUM is on. The rest of this tutorial assumes that the automatic line numbering feature is ON and that its increment value is 20. If you want an automatic increment other than 20, press (CONTROL)-(Q) to leave the editor and to return to Applesoft. Then type POKE 38925, <n> where <n> is the increment number.

(CONTROL)-(Q) Quits editor

Remember to press CONTROL-E to reenter the editor after you've specified the proper increment.

CONTROL-N means Numbering

If you don't want to use Boston Window's automatic increment feature, just press CONTROL-N while in the editor. This switches the automatic line numbering feature to OFF; the left box on the status line changes its message to MAN NUM. Try it now—but remember to reactivate automatic numbering by pressing CONTROL-N again.

Moving the Cursor Around

Boston Window provides a number of commands for moving the cursor around, and you'll find all of them described here. Of course, in order to move the cursor over characters, you first need some characters over which to have the cursor move. Assuming you've pressed RETURN to make a line number appear, type this into the editor (don't worry about any mistakes you make), but don't press RETURN again:

```
PRINT "COME LET US GO, YOU AND I,"
```

Pure Cursor Moves

You've just typed your first line of code using Boston Window. So far, it looks pretty much like plain old Applesoft. But press CONTROL-B and you'll see the cursor jump to the beginning of the display line (not the program line).

CONTROL-B means Beginning of line

Jump back to the end of the display line now by pressing CONTROL-E.

CONTROL-E means End of line

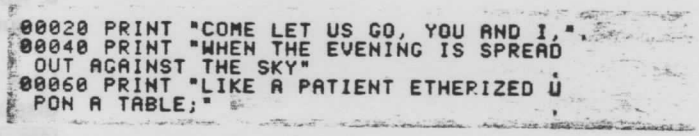
Note that these jumps are **pure cursor moves**. That is, these commands move the cursor around without affecting the text at all. Any changes to the text in Boston Window are immediately reflected on the display. What you see is what you get.

As well as making the cursor jump back and forth over whole lines, you can also use Boston Window commands to move the cursor over individual characters. Move the cursor left on your Apple IIe by pressing ←, and then right by pressing → (CONTROL-A and CONTROL-S work just as well).

← or CONTROL-A moves cursor left

→ or CONTROL-S moves cursor right

To experience the commands for moving the cursor up and down, you'll need to type in a couple more lines. Press RETURN (the cursor needn't be at the end of the line) and type two more lines of code (again, don't worry about any typos for the moment) so that your display looks like this:



Note that Boston Window typed the line numbers for you—automatic line numbering is working well! Now press ↑ (or CONTROL-W) a few times to move the cursor up the display, and then ↓ (or CONTROL-Z) to move the cursor down.

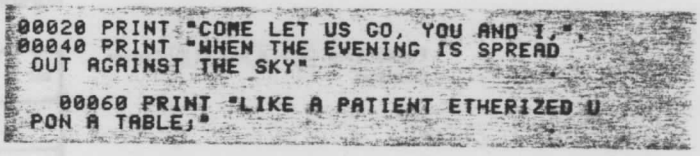
↑ or CONTROL-W moves cursor up a line

↓ or CONTROL-Z moves cursor down a line

That beeping you hear is Boston Window's way of telling you you're at the top or bottom of a program line—that is, that you've reached the upper or lower limit of the window.

Opening the Window

The term **window** refers to an invisible frame that surrounds each program line. When you begin to type a line of code, the frame is built beginning to the left of and above the line number and extending to the right of and below the last character in the program line. If you could see the frame around line 60's window, it would look like this:



You can add, delete, or change characters within the window (you'll see how in just a moment). But once you start to edit a program line, the window is closed and you're confined in your movements to the area within the window. To open the window press CONTROL-O. Once the window is open, you can move the cursor anywhere in the program.

CONTROL-O Opens the window

To see this more clearly, try this exercise:

1. Open the window by pressing CONTROL-O.
2. Use the various cursor movement keys to put the cursor over the S in SKY in line 40.
3. Now press → to move the cursor until it's over the ending quotation mark.

Look at the right box on the status line at the bottom of the display. The message reads WINDOW OPENED. Press the SPACE bar now—and watch the message change to WINDOW CLOSED. Try moving the cursor up to line 20 or down to line 60 using the arrow keys or CONTROL-W and CONTROL-Z. When you reach the window's edge, your computer beeps at you. In fact, the computer beeps any time you try to leave a closed window.

There are a number of ways to open the window so that your cursor can, in hawklike fashion, swoop around the program searching for other text to edit. The simplest way is to press CONTROL-O; you'll discover other methods as you go along. Try it now, and use the cursor movement keys (the arrows or CONTROL-W, -A, -S, and -Z) to wander around a bit.

The Cursor Control Diamond

The alert reader (you clever rascal) will have noticed two things by now. First, the keys W, A, S, and Z form a diamond on the keyboard, the four points of which symbolize the direction of the cursor move (see Figure 2e-4).

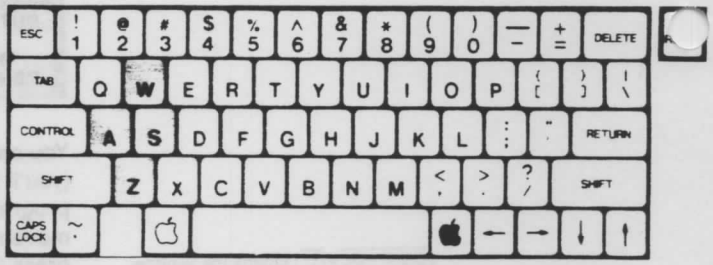


Figure 2e-4. W-A-S-Z Cursor Controls

Second, these moves are like the pure cursor moves accessed by the Apple IIe's four arrow keys, and like the I-J-K-M diamond in escape mode. In fact, escape mode does, indeed, work in Boston Window as long as the window is open (see Figure 2e-5).

Press [ESC] now and look at the center box on the status line; the message there reads ESCAPE MODE EDIT. Press the [SPACE] bar to leave escape mode, and the ESCAPE MODE EDIT message changes again (and you'll soon learn what the new message means).

escape mode cursor moves work in Boston Window

Before going on, experiment more with the arrow keys, [W]-[A]-[S]-[Z], and [I]-[J]-[K]-[M] in escape mode to move the cursor around. They all do the same thing, but some people find one method more convenient than another.

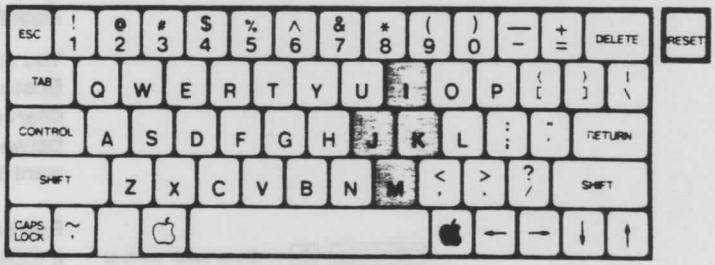


Figure 2e-5. [I]-[J]-[K]-[M] Cursor Controls

Inserting Characters

Boston Window provides some easy facilities for adding characters to the middle of a program line. Open the window with [CONTROL]-[O] and use the pure cursor moves keys to move the cursor so that it's over the comma in the first line:

```
00020 PRINT "COME LET US GOD, YOU AND I,".
```

Assuming you aren't in escape mode (and if you are, press the [SPACE] bar to leave it), type a space, followed by the word THEN. Boston Window automatically inserts the characters you type, pushing the cursor and any characters after it to the right. This is on your screen now:

```
00020 PRINT "COME LET US GO THEND, YOU A
 ND I"
```

Leaving and Reentering Insert Mode

CONTROL-@ means Overprint or move Over

Boston Window is automatically in insert mode all the time (now you know what the status line's AUTOINSERT message means). If you don't want to be in insert mode, just press CONTROL-@ (@ is SHIFT-2). Any characters you type will replace the next characters, just as would happen in regular Applesoft. Press CONTROL-@ again and you'll restore insert mode. Try it now to see what happens—and notice the change in the message on the status line. Don't worry about all the extra characters you might type; you'll soon learn how to delete them.

Inserting Control Characters

You can also embed control characters in your programs using Boston Window. But since Boston Window uses many control characters for commands, it needs to be able to distinguish between command control characters and control characters you want to include in program lines.

CONTROL-P embeds each control character

Precede each control character you want to embed in the text with a CONTROL-P so Boston Window can know what you want. Embedded control characters show in a listing as highlighted text (black letters on a white or green field). Try it now to get some practice—enter CONTROL-P CONTROL-K, for instance, to embed a CONTROL-K in the text (and, once again, note the change in the status line's middle box). Then read the next section so you can take the control character out again!

Deleting and Restoring Characters

Ordinarily, pressing the ← and → keys in the Apple IIe makes the cursor pass over characters without changing what appears on the display. But press the ← key in Boston Window while holding down the 🍎 key and characters start to disappear. At the same time, all the characters from the cursor to the end of the program line move to the left to fill the empty space.

🍎-← deletes, 🍎-→ restores characters

Press 🍎-← now three or four times to see this happen. If you practiced typing in control characters with CONTROL-P, use this deleting feature to remove them. Then press the → key a few times while still holding down the 🍎 key, and the characters reappear (they were never lost—just stored in a **buffer**). You can delete and restore up to 255 characters using the arrow keys in combination with 🍎.

It turns out that the text of line 20 is incorrect (it's supposed to be the first line from T. S. Eliot's poem "The Love Song of J. Alfred Prufrock"). Move the cursor to the space between the words COME and LET. Then use ⌫-← to erase the word COME.

Deleting Characters Under the Cursor

Just as ⌫-← deletes characters to the left of the cursor, CONTROL-D deletes the character under the cursor. Luckily there is still a minor problem with line 20, just waiting to be corrected via CONTROL-D: there shouldn't be a space between the opening quotation mark and the word LET. Move the cursor over that space and press CONTROL-D; the space disappears into the delete buffer.

CONTROL-D means Delete character under cursor

If you envision the cursor as lying to the left of the character over which it appears (sort of sandwiched between two characters), then you can think of ⌫-← as deleting the character to the left of the cursor and CONTROL-D as removing the character to the right of the cursor. Play with CONTROL-D a bit and you'll see how it differs from ⌫-←'s action.

A Brief Note About Retrieving Characters: Boston Window has several storage areas called buffers where characters are temporarily placed for safekeeping. When you delete characters from a program, they are moved into the delete buffer so that you can later retrieve them via ⌫-→.

Retrieving characters deleted via CONTROL-D, however, can have some strange results; they'll come out backwards. For instance, if you delete the word PRINT with CONTROL-D's and then use ⌫-→, you'll get TNIRP!

Deleting to End of Line

CONTROL-Y deletes to end of line

Boston Window offers a handy command for deleting text from the cursor to the end of the program line—CONTROL-Y. This command can save you lots of time when you want to remove REM statements selectively, or for those situations where you've modified a line of code and have a bunch of characters left over. Characters deleted via CONTROL-Y go into the buffer, so you can get them back if you need to.

Note to Mass REM Deleters: If you want to strip your program of all REM statements, use Applesoft Programmer's Assistant's &C command. See Chapter 1 for the details.

Deleting Whole Lines

CONTROL-L means delete Line

There's one more command you can use to delete text into a buffer; press CONTROL-L and the entire program line disappears into the delete buffer. This feature is useful if the line you've typed is so messed up that it's beyond repair. Once again, as with the other deletion commands, the line isn't really gone; it's in the buffer. You can press ⌂-→ until the line is back on the display.

▲ **Warning**

When you are using Boston Window, pressing the DELETE key deletes characters to the left of the cursor, just as the ⌂-← does; but the deleted characters do *not* go into the delete buffer.

Completing a Program Line

RETURN finishes work on a line

No matter where the cursor is in a program line, when you press RETURN the whole line goes into Applesoft's program memory. Move the cursor to the middle of line 60 now and press RETURN—none of the characters will be lost.

Actually, when you press RETURN a number of things happen. The current program line goes into Applesoft's program memory, a new line number with the appropriate increment is written to the display, and the cursor is placed so that you can begin to type the next line, with a window closed around the new program line.

Inserting New Program Lines Between the Old

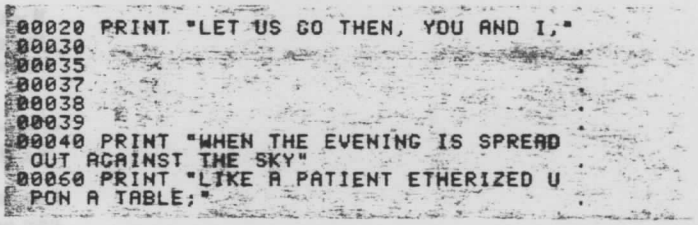
You've already seen how Boston Window's autonumbering work to type new consecutive program lines. You can also use it to type new program lines between existing ones.

Move the cursor so that it's somewhere in line 20. Then press RETURN. If autonumbering is in effect (as it should be, unless you shut it off by pressing a CONTROL-N), Boston Window types line number 30 for you and correctly places the cursor so you can type new code.

When you press RETURN, Boston Window takes the line number that it is currently on (line 20) and attempts to set up a new line number for you. Ordinarily, the new line number would be the old line number plus 20, which is the standard increment. But line number 40 is already being used; so Boston Window takes the current line number (20) and averages it with the number of the next existing line (line 40). Some quick arithmetic:

```
20 + 40 = 60
60 / 2 = 30
```

Boston Window assigns the number 30 to the next line. If you press RETURN again, Boston Window leaves line 30 blank and averages 30 with 40—and sets you up to type line number 35. Press RETURN a few more times; when Boston Window runs out of integers to use for line numbers, it just beeps at you.



If autonumbering is not in effect (that is, if you've disabled it by pressing CONTROL-N), then Boston Window just opens a space between the current and next program line and waits for you to type in your own number.

Blank Lines Are Harmless

All those blank lines on the display are harmless. You can ignore them, make them go away by pressing a few CONTROL-L's (CONTROL-L deletes a program line), or get rid of them by leaving and restarting Boston Window with CONTROL-Q CONTROL-E. Try both of these methods now, just for the practice.

Adding a Lower Line Number

Boston Window lets you add line numbers before existing ones as well as after them via the CONTROL-^ command. To see what this is all about, move the cursor so that it's at the start of line 20. You've just seen that by pressing RETURN you can type a new program line between lines 20 and 40; to type a program line *before* line 20, press CONTROL-^. Boston Window makes a space *above* line 20 (by scrolling all text down one line) so you can begin typing more code.

Notice that Boston Window doesn't assign a line number here; if you want one, just press RETURN—and you'll be typing code a line 10 (the average of 20 plus 0, since 0 is the lowest line number you can use in Applesoft).

CONTROL-^ sets up immediately preceding line

Ordinarily Boston Window sets up a line number for you when you use CONTROL-^. It didn't give a line number here because you typed a new lowest line; to Boston Window, a new lowest line is a special situation. To get a line number, either type in your own or just press RETURN.

You can use CONTROL-^ anywhere in the program. If there is no room for a new line number preceding the current one (for instance, you already have a line number 99 and you're currently in line number 100), pressing CONTROL-^ just makes Boston Window beep at you.

If you encounter a situation where you need a new line number and there isn't room for one, use the Applesoft Programmer's Assistant's renumbering feature described in Chapter 1.

A Little Practice

Leave the editor now (CONTROL-Q) and list the program so you can see your handiwork. Then run it. If you don't like how the display looks, go back into the editor by pressing CONTROL-E and fix the code. When you're through, go on to the next section to learn about the rest of Boston Window's features.

Dealing with Larger Programs

To experience the rest of Boston Window's features, you'll need to have a longer program in memory. Load (but don't run) the MAXWELL program. Then enter Boston Window (CONTROL-E).

Scrolling Through Your Program

With the window open, the cursor can zip anywhere in the program. When you reach the bottom of the display, the code scrolls up one line at a time. Try using ↓ at the display's bottom to see this happen.

CONTROL-V scrolls down through program; pressing any character key stops it

Having to hold down keys just to list through a program is somewhat tedious; to have Boston Window do it for you, press CONTROL-V. CONTROL-V moves the cursor to the bottom of the display and scrolls the program continuously until the listing's end, much like Applesoft's LIST command. To stop the listing at any point, press any character key. To control the listing, experienced users of Boston Window keep the CONTROL key pressed with one finger, using another finger to press the V.

To Use or Not to Use CONTROL-C: When you are listing an Applesoft program, you would ordinarily use CONTROL-S to pause and continue the listing; if you wanted to terminate the listing, you would use CONTROL-C. Such is not the case with Boston Window's CONTROL-V scrolling command. When you stop the scrolling by pressing a character key, the scrolling is in fact terminated. There is no need for a CONTROL-C; in fact, Boston Window uses CONTROL-C to start a special Copy Characters sequence (see below). If you accidentally press CONTROL-C and strange things start appearing on your display, press CONTROL-C again followed by CONTROL-O, and everything will be OK.

CONTROL-T scrolls towards the Top; pressing any character key stops it

Unlike plain vanilla Applesoft, Boston Window also allows you to scroll through a program backwards. Press CONTROL-T and Boston Window places the cursor at the top of the display and begins scrolling until it reaches the start of the code. As with CONTROL-V, any character key stops the scrolling and another press of CONTROL-T starts it again.

In Case You Wondered About Highlighted Characters: In scrolling through MAXWELL, you will have noticed all those highlighted characters. Each one is an embedded control character. It happens that most of them are special commands used by the High Resolution Character Generator, described in detail in Chapter 6.

Skipping Through Your Program

Boston Window provides a quick way of moving from one section of a program to another: it's called the GOTO command.

CONTROL-G lets you go anywhere in the program

Assume that you want to edit line number 2000. Press CONTROL-G and Boston Window responds by displaying GOTO: in highlighted text near the top left of the display. Now type the number 2000 and press RETURN. Boston Window obediently shows a displayful of code beginning with line 1990 (the line of code immediately preceding the line specified), with the cursor positioned over the first character of line 2000.

Of course, now that you're at line 2000 you don't have to edit it; you can use GOTO just to look at different sections of code.

If the line number you specify with CONTROL-G doesn't exist, then the cursor goes to the next higher numbered line. If the specified line is higher than the final line number in the program, then the cursor goes to the program's final line. If the line number you specify is lower than the lowest line, then the cursor goes to the first line of the program.

A Quick Zip Tip: To get to the first line of the program, type

CONTROL-G RETURN

To get to the last line, type

CONTROL-G 63999 RETURN

Finding a String

You can also have Boston Window find a particular string of characters located anywhere in the program. Press CONTROL-F followed by any string of characters (up to 34 characters long), and press RETURN. Boston Window moves the cursor to the first occurrence of the string beyond the cursor's present position.

CONTROL-F Finds a string

Assuming the cursor is at the start of line 10 in the current program (and if it isn't, put it there by typing CONTROL-G 10), press CONTROL-F. `FIND:` appears in highlighted text just below line 10. Now type the word `WAVE` and press RETURN. The cursor moves down the display to line 200, just beyond the word `WAV`.

If you press CONTROL-F again, Boston Window tries to find the next occurrence of the word `WAVE` following the cursor's current position (FIND always searches ahead). It turns out that the word doesn't appear again in the program; so after trying its best to find it, Boston Window signals by beeping that it can't find a new `WAVE`.

FIND does a literal search for its strings. That means that if you tell Boston Window to FIND the letters `IN`, it will stop at the `IN` in `INC.`, `INITIALIZE`, and so on. If you want to find *only* the whole word `IN`, then you must specify the spaces separating the word from its neighbors. Further, if you specify `IN` (uppercase characters), then FIND won't discover lowercase `in`. To the FIND command, uppercase is uppercase and lowercase is lowercase. You must search for each separately.

To Search for Another String

If you've found the string you're searching for and you want to give FIND another string to search out, press CONTROL-G RETURN. This puts the cursor at the start of the first line, with FIND ready for new material. If you press CONTROL-F now and just press RETURN, the old material in the buffer is retained and Boston Window looks for the most recently specified string.

Searching for a Control Character: When using CONTROL-F to find a control character, you must press CONTROL-P before typing the control character for the search.

A Wildcard Character

CONTROL-W means Wildcard

Typing CONTROL-W as a character in the FIND string enters a highlighted blank space in the string. The highlighted blank space is a wildcard character, matching any character it finds in that position in the string. You can have as many wildcard characters in a string as you want. Try this:

What You Type	What Happens
1. Press CONTROL-G	
2. Press RETURN	Moves cursor to top of program
3. Press CONTROL-F	
4. Type WA and press CONTROL-W CONTROL-W	Tells Boston Window to look for any string at least four characters long starting with characters WA
5. Press RETURN	Begins the search; stops at end of WALK in line 100
6. Press CONTROL-F	Continues search; stops at end of WAVE in line 200

CONTROL-R means Replace

Replacing a String

Using CONTROL-R, a variation of the FIND command, you can replace some or all of the occurrences of a specified string with another string.

Replacing Selectively

In the following example you'll change the word GOSUB to GOTO (to Boston Window, a command word is just another string). Here's what to do:

What You Type	What Happens
1. Press CONTROL-G	
2. Press RETURN	Moves cursor to top of program
3. Press CONTROL-F	
4. Type GOSUB	Tells Boston Window to look for the string GOSUB
5. Press CONTROL-R and type GOTO	Tells BW to replace GOSUB with GOTO
6. Press RETURN	Cursor moves to space following first occurrence of the word GOSUB (line 40)

Now you have a choice. You can press another CONTROL-F, leaving the GOSUB in line 40 intact and causing the cursor to mo to line 80's GOSUB. Instead, press CONTROL-R, and *voila!* the GOSUB becomes a GOTO.

You can continue to press CONTROL-F (to find the string again) or CONTROL-F CONTROL-R (to both find and replace the string) until you go through all occurrences of GOSUB in the program.

Replacing Globally

Boston Window offers you a final variation on FIND: the global replace option. It lets you replace all occurrences of a given string (or any group of characters) in a program with a single command. Try this:

1. Press [CONTROL]-[G]
2. Press [RETURN] (gets you to start of program)
3. Press [CONTROL]-[F]
4. Type REM
5. Press [CONTROL]-[R]
6. Type RAM (but don't press [RETURN] yet)

[CONTROL]-[G] replaces Globally; pressing any character key stops it

You're going to replace all occurrences of REM with RAM—but instead of pressing [RETURN], press [CONTROL]-[G]. In this context, [CONTROL]-[G] stands for Global Replace. The cursor zips through the program, changing before your very eyes all REMs to RAMs.

If you change your mind you can stop this wholesale replacing while it's still going on by pressing any character key; Boston Window signals that it understands your cease-and-desist command by beeping at you. But any changes that Global Replace made up to the time you stopped the process remain in effect.

To replace all occurrences of a given string with the null string—that is, a string with no characters—follow the [CONTROL]-[R] command directly with [CONTROL]-[G].

Copying Code or Strings

Quite often in programming you come across situations where you need to use the same code or strings again and again, but for one reason or another you don't want to use subroutines. To reduce the drudgery of repetitive typing, Boston Window offers the copy buffer.

[CONTROL]-[C] means Copy or Capture

Use [CONTROL]-[G] (which means GOTO in this context) to move the cursor to line 910. Now press [CONTROL]-[C] and use [→] (actually any character key will do) to move the cursor until it's over the semicolon. As the cursor moves, the characters over which it passes change to highlighted text to let you know that they have been copied into the copy buffer (look at the status line; it, too, tells you when COPY is operating). When the cursor is over the semicolon, press [CONTROL]-[C] again; the copied characters now appear in normal video, and the copy buffer is closed.

If you go clear to the end of the program line, the copy buffer closes itself. The buffer holds up to 255 characters.

CONTROL-I or TAB means Insert

Now move the cursor to line 1100 and press RETURN to set up a new line. Then press CONTROL-I and the contents of the copy buffer (line 910's code without the semicolon) are inserted into line 1105. Pressing TAB means the same thing as CONTROL-I in Boston Window.

The buffer isn't empty; it still holds the characters you just put into it. To prove it, do this now:

1. Move the cursor to line 60.
2. Use CONTROL-E to move to the end of the line.
3. Add a colon (:).
4. Press CONTROL-I again.

The line now looks like this:

You can use the contents of the copy buffer again and again; it won't change until you issue another CONTROL-C command.

Half-Screen Mode for Debugging

CALL 34689 gets you half-screen mode

Boston Window's final feature is half-screen mode. By issuing the Applesoft command CALL 34689 you can make debugging easier by watching the program run in the bottom half of the display while the code is visible in the top half.

Turning Off the Status Line: You can't turn off the status line while the program is in half-screen. Return to full-screen by typing CALL 34698 from Applesoft (CONTROL-Q puts you back in Applesoft). Then type POKE 38638, 96 to turn off the status line.

Half-screen functions a little differently from full-screen. To see it work, first type NEW from Applesoft. Then, using autonumbering and the copy buffer to make the job quick and easy (see the box below if you're having trouble), type the following short program while you're still in full-screen mode:

```
00020 FOR X=1 TO 1000
00040 PRINT X;
00060 NEXT X
00080 GOTO 20
00100 REM
00120 REM
00140 REM
00160 REM
00180 REM
00200 REM
00220 REM
00240 REM
00260 REM
00280 REM
00300 REM
```

About That Shortcut: In case you're having trouble figuring out how to use the copy buffer to type in the sample program, here's how to do it.

1. Type the first four lines as you normally would, using the automatic numbering feature to type the line numbers for you. If this feature isn't working, press (CONTROL)-(N) to turn it on.
2. When you get to line 100, type REM but don't press (RETURN).
3. Move the cursor to the R in REM.
4. Press (CONTROL)-(C) to start putting characters into the copy buffer.
5. Press the (SPACE) bar (or any character key) to copy the word REM.
6. Press (CONTROL)-(C) again.
7. Press (RETURN) and then (CONTROL)-(I).

As you can see, the last step in this series made line number 120 appear, with REM typed after it. Repeat step 7 a few more times and you're done.

CALL 34698 restores full-screen

Now leave the editor via (CONTROL)-(Q) and issue a CALL 34698 to set half-screen mode. As soon as you type this command and press (RETURN), the display flashes and only lines 20 through 240 are visible; that's all that will fit on the top half of the display.

Now run the program. The bottom half of the display quickly fills up with numbers. When you're ready, press CONTROL-C to s the action; then get into the editor via CONTROL-E.

Switching between CONTROL-V (scroll down) and CONTROL-T (scroll up) will show you that all the code is still there; it's just somewhat hidden.

To restore full-screen again, type `CALL 34698` from Applesoft (use CONTROL-Q to leave the editor and get back to Applesoft).

▲ **Warning**

Don't use Applesoft's TEXT command to reset the display window from half-screen to full-screen. TEXT does funny things to Boston Window, making it in turn do strange things to the system memory, and ultimately causing everything to go someplace indeterminate where it might never, never be found.

To recover from TEXT, type `CALL 34689` to restore the half-screen display; or type `CALL 34698` to set up full-screen display.

Section 2

Boston Window Reference

This section contains information you'll want to refer to from time to time as you use Boston Window. It covers the material discussed in the tutorial section but in a more condensed form; you'll find this part of the chapter especially useful as an expansion of the notes from the Quick Reference Card at the back of the manual.

Additionally, there are comments here on the general structure of Boston Window, hints for troubleshooting unusual problems that might pop up, and special options for more advanced programmers.

Boston Window and Line Numbers

When Boston Window begins editing a line of code, the line is first deleted from Applesoft's program memory. When editing is completed the program line is inserted into program memory. Therefore, it's extremely important that you type program line numbers correctly. Most people type line numbers in the manner about to be described anyway; these paragraphs are included just to be on the safe side.

The editor assumes that the line number begins at the display's left margin. The editor also assumes that the line number is contained in the first five columns of the display line. It reads up to five characters or until it reads a nonnumeric character, whichever comes first.

In order for a line number to be valid it must follow the same rules that apply to line numbers in Applesoft BASIC, with three additional constraints:

- The line number must start in the left margin (column 1).
- The line number cannot have any spaces in it.
- The line number must be contained in the first five columns.

Warning

Never change a line number to an existing line number. If you do, the original line of code is destroyed and replaced by the new one. Further, if you attempt to swap the positions of two or more program lines by changing the line numbers on the display, you put yourself in danger of losing *all* the lines you're trying to reposition.

Troubleshooting Boston Window

Sometimes Boston Window might act in ways that surprise you. In the vast majority of cases, you can trace the strange behavior to some command you have inadvertently given or to some function you haven't used enough to understand fully. Here are some such confusing situations you might come across, with suggestions for clearing the problems up.

If You Want to Initialize a Disk...

Be sure the SYSTEM MASTER version of DOS is in memory. Initializing a blank disk while Boston Window is hidden in high memory—between DOS itself and the file buffers—will result in a nonstandard copy of DOS being written on the disk. As a result, there is a reduction of available memory to your application program.

If You Try Using Boston Window with Integer BASIC...

You can execute Boston Window from Integer BASIC, but the program won't work on Integer code. Once you've got Boston Window in memory, however, you can load an Applesoft progra and Boston Window will work fine.

If Nonsense Characters Fill the Screen ...

If you get a screen full of **garbage** when you enter the editor, you probably have somehow gotten yourself into the Monitor or Integer BASIC. If this happens, press CONTROL-Q to leave the editor and type FP to get into Applesoft. If you type FP and nothing happens, you'll have to restart the system. Life is filled with such minor disappointments; hang in there.

If Boston Window Seems to Disappear...

Boston Window occasionally loses its connection to the rest of the system (for instance, when you press (CONTROL)-(RESET)) and seems to vanish into The Big Buffer In The Sky. If this happens, type `CALL 34698` from Applesoft and Boston Window is usually accessible again.

If Random Program Lines Appear...

Random lines at the display's bottom means that Boston Window has been partially disconnected from the system, probably because you issued a PR# command. Just type `CALL 34698` from Applesoft to reconnect.

If You Use Global Replace...

Replacing globally can be a dangerous practice. Be sure that you really want to replace *every* occurrence of a given string before using this feature—and even then, stand ready to halt the process by pressing the (SPACE) bar (or any other character key). Also remember that, if you change a word like *in* to *out*, you might end up with strange *out* formation in your file! Be sure to indicate whole words by including spaces before and after the characters you want changed.

If You Use Half-Screen Mode...

Don't use Applesoft's TEXT command to reset the screen window from half-screen to full-screen. To recover from TEXT, type `CALL 34689` to restore the half-screen display; or type `CALL 34698` to set things up for full-screen editing.

If Your Program Has Lines Longer than 239 Characters...

There are special Applesoft utility programs available that allow you to type programs with lines longer than 239 characters. If the program you are editing with Boston Window has been written using such a utility, and if you attempt to edit a line containing more than the normally allowed number of characters (239 including embedded spaces), the line will be destroyed.

Before editing such a line, break the line into shorter segments from Applesoft rather than from Boston Window. Then you can safely enter the editor and change away. For other ways of handling long lines, see the next section, on **packed lines**.

Special Packed Line Options

Some programmers like to save memory space by combining several statements on the same program line. If you are one of those programmers, then you know the frustration of trying to change an element in a long program line. Quite often you'll be copying the line into memory, having made your changes, when Applesoft begins beeping at you—letting you know your line now has more than the 239-character limit. Boston Window gives you three ways to beat this numbers game, in the form of three different POKE commands.

Reducing "PRINT" to "?"

Boston Window's first byte-saver changes all PRINT statements to the familiar question mark (?), saving four characters for each instance of use. The command, issued from Applesoft, is `POKE 37781, 186`. To restore the normal display, type `POKE 37781, 0`.

`POKE 37781, 186` displays `PRINT as ?`

Deleting All Extra Spaces

The second POKE command in this series does away with embedded spaces in a program line. Normally, the editor sends program lines to Applesoft as they appear on the display; thus any embedded spaces count against the 239-character total. But with `POKE 37048, 160` you instruct the editor to delete all spaces not in a quoted string as a line is sent to Applesoft. In a line of, say, 50 words, that's a savings of 49 characters! To restore normal operations, use `POKE 37048, 0`.

`POKE 37048, 160` eliminates nonquoted spaces

Note that Boston Window takes out the extra spaces only as it sends a line to Applesoft and not while the editor is displaying the line—you won't see any changes on the display. A word of warning here, though: unless you enclose the text in quotation marks, the spaces between words in REM statements are also eliminated, resulting in virtually unreadable comments. The solution is either to put all the text of REM statements in quotation marks, or to use the third special POKE command.

Making the Display Compact

The final compacting method leaves spaces in REM statements and quoted strings, but takes out spaces between command words in the display. You'd use this method, which you access with `POKE 37789, 186` and cancel with `POKE 37789, 0`, when your program line takes up more than seven display lines.

`POKE 37789, 186` compacts the display

Dense program lines require more screen lines to display than do normal program lines, and this length is amplified by the editor's display formatter, which adds extra spaces for clarity. In order to accommodate the size of the buffers the editor uses for scrolling, the editor sets a limit of seven on the number of screen lines a single program line can occupy.

If a program line scrolls onto the screen and is longer than seven screen lines, the editor truncates the screen display; the copy of the line in program storage is unaffected. Issuing `POKE 37789, 186` changes the way the editor formats the display, so that even the longest program line fits in seven lines.

Using this method, however, jams all the command words next to each other in the display, making program lines very difficult to read. Unless you are writing *very* long lines and are concerned with seeing the entire line on the display, it makes more sense to use the first two POKE options—and remember to put your REM text in quotation marks.

Quote those comments!

Command Summaries

Presented below are lists of Boston Window's commands, categorized in a number of different ways. First, the commands are grouped as to functions—cursor movement, deletion, and so on. Later is an alphabetical list of all control character commands. Finally you'll find a diagram showing Boston Window's many cursor movement commands. You'll also find these summaries on the handy tear-out reference card at the back of this manual.

Functional Command Summary

Editor Access

Enter editor	CONTROL-E
Quit editor	CONTROL-Q

Cursor Movement

Open window	CONTROL-O
Move left one character	← or ESC J or CONTROL-A
Move right one character	→ or ESC K or CONTROL-S
Move up one line	↑ or ESC I or CONTROL-W or -K
Move down one line	↓ or ESC M or CONTROL-Z or -J
Move to left edge of screen	CONTROL-B
Move to right edge of screen	CONTROL-E
Scroll toward top of program	CONTROL-T
Scroll toward bottom of program	CONTROL-V

Delete

Delete character left	CONTROL-H or ◁-←
Delete character at cursor	CONTROL-D
Delete program line	CONTROL-L
Delete to end of program line	CONTROL-Y

Restore

Restore last character deleted	CONTROL-U or ◁-→

Copy

Open, close copy buffer	CONTROL-C
Insert copy buffer contents	CONTROL-I or TAB

Find, Replace

Find specified string	CONTROL-F
Add wildcard character	CONTROL-W
Replace string with specified string	CONTROL-R
Global Replace (after CONTROL-F CONTROL-R)	CONTROL-G

Start New Line

Start new higher line	CONTROL-M or RETURN
Start new lower line	CONTROL-^

Miscellaneous Functions

Autonumbering ON-OFF	CONTROL-N
Autoinsert ON-OFF	CONTROL-@
GOTO specified line number	CONTROL-G
Insert control character	CONTROL-P

Alphabetic Command Summary

CONTROL-A	Moves cursor left one character—same as ESC J
CONTROL-B	Moves cursor to left edge of current display line
CONTROL-C	Opens/closes copy buffer
CONTROL-D	Deletes character at cursor position, moves all text left one space
CONTROL-E	From Applesoft, enters editor; from editor, moves cursor to right edge of current screen line
CONTROL-F	FINDs specified string
CONTROL-G	GOTO specified line number; at end of CONTROL-F CONTROL-R sequence, does Global Replace

CONTROL-H	Deletes character to left of cursor, moves all text left one space—same as ⌂-←
CONTROL-I	Inserts contents of copy buffer at cursor position
CONTROL-J	Moves cursor down one line—same as CONTROL-Z, ESC M
CONTROL-K	Moves cursor up one line—same as CONTROL-W, ESC I
CONTROL-L	Deletes program line containing cursor
CONTROL-M	Opens window on current program line, closes window on new line; if autonumbering is in effect, assigns line number to new line higher than current line—same as RETURN key press
CONTROL-N	If autonumbering is ON, turns OFF; if autonumbering is OFF, turns ON
CONTROL-O	Opens window
CONTROL-P	Use before adding control character to text; control character displayed in inverse video
CONTROL-Q	Leaves the editor and returns to Applesoft with cursor at bottom left of screen
CONTROL-R	Used after CONTROL-F to establish replacement string or to effect the replacement
CONTROL-S	Moves cursor right one character—same as ESC K
CONTROL-T	Moves cursor toward top of window or scrolls to start of program; stopped by press of any character key
CONTROL-U	Restores one character from delete buffer—same as ⌂-→
CONTROL-V	Moves cursor toward bottom of window or scrolls to end of program; stopped by press of any character key
CONTROL-W	Moves cursor up one line—same as CONTROL-K, ESC I; wildcard character in FIND string
CONTROL-X	Cancels current command (FIND or GOTO)
CONTROL-Y	Deletes into buffer all characters from cursor to end of program line
CONTROL-Z	Moves cursor down one line—same as CONTROL-J, ESC M
CONTROL-@	If autoinsert is ON, turns OFF; if autoinsert is OFF, turns ON
CONTROL-^	Opens window on current line; closes window on line to have lower line number than current line; if autonumbering is in effect, assigns line number to new line lower than current line

Special CALLs and POKEs

Reconnect Boston Window, full screen	`CALL 34698`
Reconnect Boston Window, half screen	`CALL 34689`
Set AUTONUM to stipulated increment	`POKE 38925, <n>`
Delete nonquoted spaces (buffer)	`POKE 37048, 160`
Restore to normal	`POKE 37048, 0`
Delete nonquoted spaces (screen)	`POKE 37789, 186`
Restore to normal	`POKE 37789, 0`
Display PRINT as ?	`POKE 37781, 186`
Restore to normal	`POKE 37781, 0`
Turn off status line	`POKE 38638, 96`
Display status line	`POKE 38638, 152`

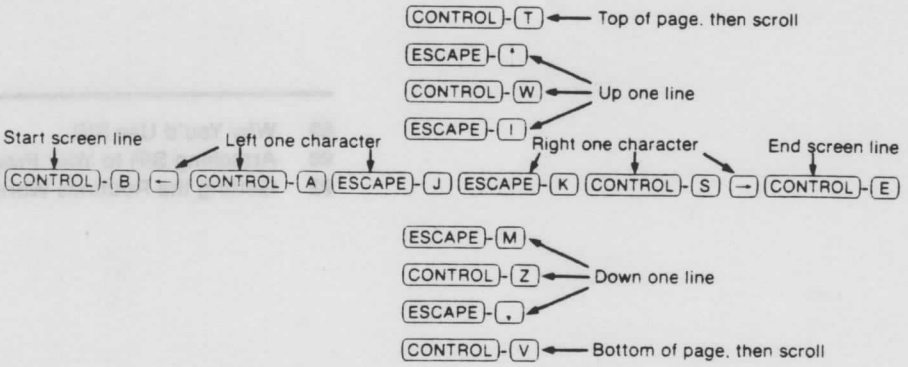
Functions of Special Keys

DELETE	Deletes character left (does *not* go into buffer)
ESC	Turns on escape mode
RETURN	Opens window on curent line, inserts blank line below current line, and closes editing window around blank line
↑	Moves cursor up one screen line
↓	Moves cursor down one screen line
←	Moves cursor left one character; with ó, removes character left and puts into delete buffer
→	Moves cursor right one character; with ó, restores last character put into delete buffer
TAB	Inserts copy buffer into text—same as CONTROL-I

Cursor Moves Diagram

Here are Boston Window's pure cursor move keys. Pressing ESC turns on the familiar escape mode; see the *Applesoft BASIC Programmer's Reference Manual* for details.

Figure 2e-6. Boston Window Cursor Moves Diagram



Page 0 Memory Usage

The following table shows the hexadecimal (HEX) and decimal (DEC) memory locations on page 0 used by Boston Window. Where two bytes are named, low byte is listed first.

HEX	DEC	PURPOSE
E3	227	Horizontal cursor position
EB	235	Vertical cursor position
EC-ED	236-237	Cursor screen address
EE-EF	238-239	Line number holding cursor
F9-FA	249-250	Address of last keyword read
FB-FC	251-252	Screen address for scrolling
FD-FE	253-254	Address of program line holding cursor

Boston Window is positioned in memory just below DOS, with DOS's input and output buffers shifted down to accommodate it. Boston Window occupies about 5 kilobytes from $8781 (decimal 34689) to $9CFF (decimal 40191); it sets the highest address for DOS buffers at $8780 (decimal 34688).

System Identification Routines

System Identification Routines

Experienced programmers know the value of taking advantage of all the features a computer has to offer. For example, if a computer can display 80 columns instead of just 40, you can write far more interesting programs if you make use of the extra display space. The first step is knowing what the computer can do.

System Identification Routines (or SIR for short) is a series of machine language routines that enable your software to distinguish among the various models and configurations—and therefore capabilities—of the Apple II family of computers. SIR will tell your program

- whether or not the machine in which it's running is an Apple IIe
- if the computer has an Apple 80-Column Text Card or not
- whether or not the computer has an Apple Extended Memory Card

Using SIR with Non-Apple Products: These routines are meant to identify Apple products. The routines are *not* designed to identify equipment produced by other companies. SIR might work for non-Apple equipment, but don't rely on it without checking carefully.

Why You'd Use SIR

If the only system for which you write software is your own, then you can skip this chapter; after all, you already know what equipment is in your own machine. But if you write software that will run in somebody else's Apple, then SIR can come in very handy. Among other things, you can use the information these routines provide to:

- make sure the equipment being used is appropriate to your software
- have the software decide whether to use certain keys and features present on the IIe but not present on the II and II Plus

- enable the program to determine whether to format text for 40- or 80-column display
- decide whether the program could take advantage of extra memory

Attaching SIR to Your Programs

SIR appears in this package in two guises. The pure assembly language version is on the DOS Programmer's Tool Kit Volume I disk in a binary file called A//E ASSEMBLY ID; a BASIC language version is on the same disk, contained in an Applesoft program called A//E BASIC ID. Thus, there are two ways to attach SIR to your programs.

The first way is to copy the binary file called A//E ASSEMBLY ID onto the disk that will hold your program, and early in your program add the line

using A//E ASSEMBLY ID

```
PRINT CHR$(4); "BLOAD A//E ASSEMBLY ID"
```

If you're going to use this method, feel free to skip down to the section called Getting the Routines Working.

The second way isn't as straightforward, but it lets you combine all the code you need in one BASIC program. It involves attaching lines 0 through 140 of the A//E BASIC ID program as a subroutine to your own program using APA's Renumber and Merge commands (see Chapter 1) to do the work. Here's how to do it.

First, have APA running in your computer. Then do this:

1. Load A//E BASIC ID (found on the disk labeled DOS Programmer's Tool Kit Volume I).
2. You only need lines 0 through 140—delete the rest.
3. Renumber the lines in some way appropriate to your program —most programmers would number them high (above 60000) and use a GOSUB to refer to them.
4. If you're going to add these lines at the end of your program as a subroutine, remember to add a RETURN statement.
5. Issue APA's Hold command (&H—see Chapter 1).
6. Load the program to which you want to add SIR.
7. Issue APA's Merge command (&M).

8. If SIR is a subroutine, add a GOSUB statement.
9. Save the program.

But you're still not quite done; read on to find out how to get the routines working.

How the A//E BASIC ID Program Works: A//E BASIC ID was written to demonstrate how to use the I.D. routines effectively in a BASIC program. The routines that do the actual identifying are written in machine language. So even though you use a BASIC program to get the routines into memory, those routines must somehow end up as machine language. If you want to see how this gets done, load A//E BASIC ID into your computer and list it:

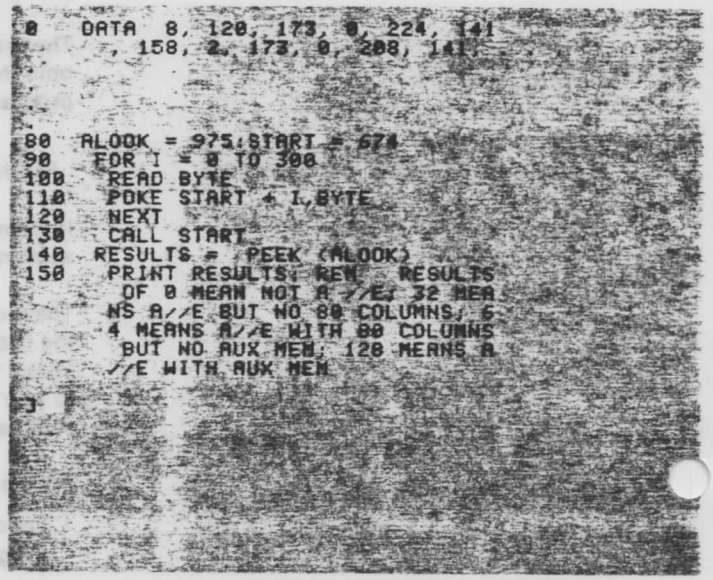


Figure 3-1. Partial Listing of A//E BASIC ID Program

Lines 0 through 70 contain, in DATA statements, the numeric equivalents of the machine language routines from the A//E ASSEMBLY ID program. When lines 80 through 120 of A//E BASIC ID are executed, the numbers in the DATA statements are entered into memory; the end effect is the same as if you had loaded the A//E ASSEMBLY ID program. Tricky, no? More in the next chapter about how to create programs like A//E BASIC ID.

Getting the Routines Working

Whether you use A//E ASSEMBLY ID or A//E BASIC ID to attach SIR to your programs, getting the routines to work is a simple matter:

- To make the routine work, issue a `CALL 674`.
- To get the information you need, issue a `PEEK (975)`

When you execute the identification routines by issuing `CALL 674` from someplace in your program, they place the information that you need into memory location 975. You need to interpret the number you find there to get any use out of it; the following table tells you how:

Table 3-1. What the Number in Memory Location 975 Means That's all there is to using SIR!

If This Is the Number	Then the Computer Is
0	not a IIe
32	a IIe with no extra memory, no 80 columns
64	a IIe with no extra memory, 80 columns
128	a IIe with extra memory, 80 columns

From Machine Language to BASIC

From Machine Language to BASIC

A program called FROM MACHINE LANGUAGE TO BASIC resides on the disk labeled DOS Programmer's Tool Kit Volume I. This program converts the contents of any binary (that is, machine language) file into a list of numbered DATA statements. You can then take these DATA statements and attach them to an Applesoft program, just as the author of SIR did (described in the previous chapter).

This program can come in very handy when you want to have one BASIC program where you might ordinarily need several programs in both BASIC and machine language.

Using the Program

Using FROM MACHINE LANGUAGE TO BASIC is quite simple; it prompts you every step of the way. To run it, first make sure that you have handy the disk with the binary file you want converted; then run the program.

Tell the program at its prompting whether you want to create a BASIC file or a text file. If you tell it you want a text file, it essentially creates a BASIC program and stores it in a text file; later, you can EXEC the file into your own program (see the DOS manual to learn about EXEC). If you tell it you want a BASIC file, it produces a short BASIC program you can add to your own programs later using APA's Renumber and Merge commands (see Chapter 1).

Whatever choice you make, the program asks you what you want to name the file. Next, it asks you for the name of the binary file you want to convert. Finally, it asks you what line number you want the DATA statements to begin with. Whatever you tell it is OK; you can always use APA to change it.

Then make sure that the disk with the binary file to be converted is in the drive. The drive whirs and clicks as the program gets what it needs. Be patient here; if the binary file you are converting is particularly long, it takes a while. The wait is worth it, though.

When the Applesoft prompting character (]) and the cursor return, catalog the disk; there you'll find listed the file you just created.

Getting the DATA Lists into Memory

Of course, converting the machine language to DATA items is only half the task. You still need a way for your program to reconvert the DATA items into machine language code. There are two quick methods you can use to store the DATA items in memory, where the computer will recognize the data as components of a machine language program.

But first, you need to know the starting address in memory where the machine language program is usually stored. You also need to know the entry point to the routine—that is, the address you would ordinarily use as the argument to a CALL command to set the routine into action. Usually, the entry point is the same as the memory starting address.

To find the address of any machine language program, first BLOAD the program. Then type whichever of the following is appropriate, depending on the memory size of your machine.

finding memory locations

If you have 48K or more of memory:

```
PRINT PEEK (43634) + PEEK (43635) * 256
```

If you have 32K of memory:

```
PRINT PEEK (27250) + PEEK (27251) * 256
```

If you have 16K of memory:

```
PRINT PEEK (10866) + PEEK (10867) * 256
```

Using FOR/NEXT to Do the Job

Lines 80 through 130 of the A//E BASIC ID program listed below demonstrate the first way to store DATA items in memory, using a FOR/NEXT loop and a POKE statement. Here's how the program's author figured out how to do it.

The author knew that the machine language routine he needed to store would begin at location 674, and that the entry point to the routine was the same as the starting address. So he set up a variable called START with the value 674.

He knew that he'd have to store one number in each address beginning at START and ending wherever the last item of DATA went. He had to find out how many DATA items there were in order to use a FOR/NEXT loop. Being a clever sort, he realized that FROM MACHINE LANGUAGE TO BASIC puts exactly forty DATA items in each DATA list; multiplying the number of full DATA lists (7) times 40 and then adding to it the number of separate items in the last list (21), he came up with 301. Thus he knew that there were 301 DATA items, each to be stored in a separate location in memory. Finally, he wrote the code:

①Sets up variables for later use
②Calls first item #0, last item 301 - 1
③Gets value of DATA item
④Puts appropriate value into correct memory address
⑤Increments I so that BYTE can go into proper location, I locations past START
⑥Executes machine language routine just stored in memory

```
①80 ALOOK = 975:START = 674
②90 FOR I = 0 TO 300
③100 READ BYTE
④110 POKE START + I,BYTE
⑤120 NEXT
⑥130 CALL START
```

Using the Automated Method

The trouble with the above method, however, is that you have to do some arithmetic (counting DATA statements, multiplying, counting each item in the final DATA list, and so on). Those of us who are totally committed to automation resent this pretechnical procedure, and would rather have the computer do the counting for us. Using the same program as an example, we would make the following changes:

Ⓐ Adds a new final DATA item greater than any in lists (all lower than 256: see note below) as an End of DATA signal

① Same as original

② Initializes variable I

③ Same as original

④ If value of BYTE > 255 then this DATA item must be the End of DATA signal; branches out of routine

⑤ Same as original

⑥ Increments counter

⑦ Gets next DATA item

⑧ Same as original

```
Ⓐ75 DATA 999
①80 ALOOK =975:START = 674
②90 LET I = 0
③100 READ BYTE
④105 IF BYTE > 255 THEN 130
⑤110 POKE START + I,BYTE
⑥115 I = I + 1
⑦120 GOTO 100
⑧130 CALL START
```

About Line 75: The highest number that can go into any single memory location using machine language is 255 (hexadecimal ff). Thus you can use any number higher than 255 for your End of DATA signal. The number 999 is arbitrary.

Whatever method you use, be assured that once your Applesoft program has sent the proper data to the proper locations, what you end up with is a real, live machine language program; and it's ready to run with a CALL to the proper address just as if you had issued a BLOAD command to a disk drive.

Animatrix

Animatrix

Animatrix is a high resolution character set editor. You use it to create fancy English letters, non-English character sets, and character graphics for animation. The disk with Animatrix on it—DOS Programmer's Tool Kit Volume I—has a number of character sets that demonstrate all three uses; this chapter will go over some of those sets as it teaches you how to use Animatrix.

▲ **Warning**

Animatrix won't work correctly with the 80-Column Text Card active. Random characters may appear on the display, the text window might inexplicably be set to a width of one column, and other unpredictable events might occur. If you are using an Apple IIe with an 80-column card, be sure to turn it off before starting Animatrix.

Starting the Program

Begin by making sure that you have properly installed the hand controls; you'll need them to work with Animatrix. Then type FF to reset HIMEM: to its highest possible position. Then type RUN ANIMATRIX.

The drive makes its customary noises and, after a while, this title page appears:

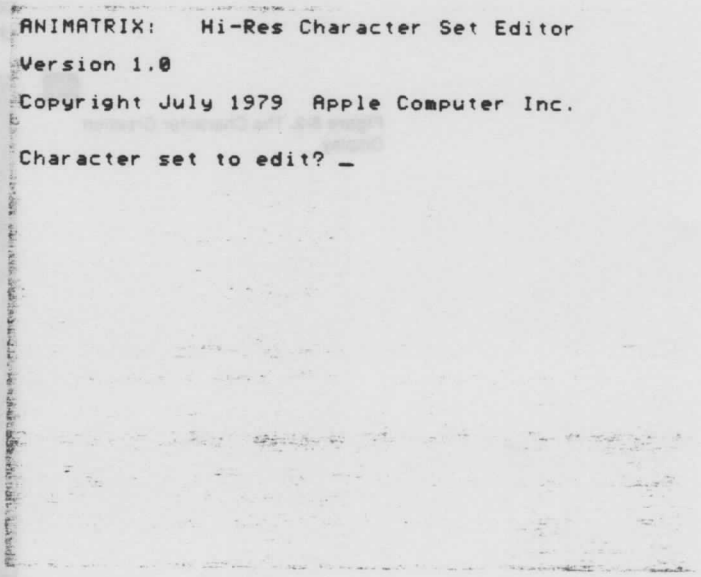


Figure 5-1. Animatrix Title Page

For Technotrolls: Animatrix loads and activates the Relocating Loader, which in turn resets HIMEM: and performs other wonders. See Chapter 7 for the details.

If you are running this program on an unmodified Apple II, you'll notice that you have lowercase characters on the display; if you're running either a II or a IIe, you'll notice that the cursor is a blinking underline. These are both signs that Animatrix and HRCG (see Chapter 6) are running correctly.

In response to the question that appears on the display, answer `GOTHIC.SET` and press RETURN.

Warning

Be sure you type the name of the character set correctly. If you misspell it, you'll find yourself out of Animatrix and back in Applesoft!

The Character Creation Display

The drive whirs again as Animatrix retrieves the character set called GOTHIC.SET from the disk. Then Animatrix draws its character creation display; it should look like this:

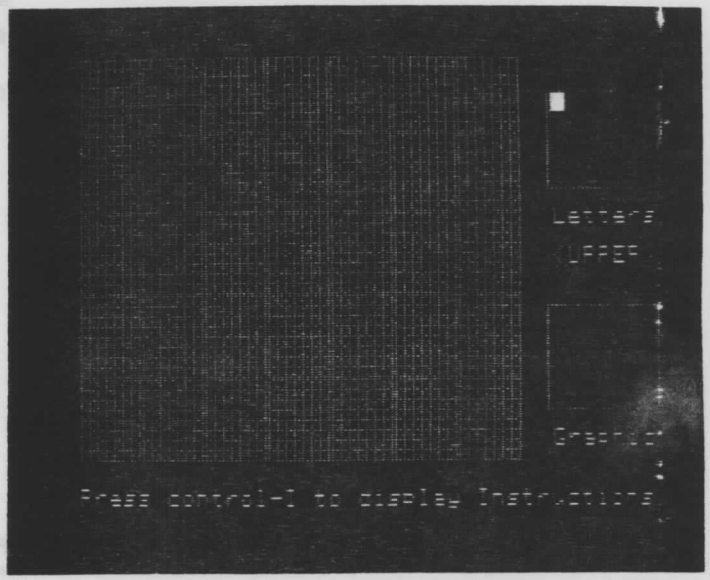


Figure 5-2. The Character Creation Display

To see what this display is all about, type the following letters slowly, waiting for the cursor to reappear after you type each letter:

```
GOTHIC
```

Here's what should have happened. First, the letter G appeared in the upper left corner of the box marked Letters. Next, the character G appeared in Gothic script in the upper left corner of the grid. Finally, a smaller (and smoother) representation of the Gothic G appeared in the upper left corner of the box marked Graphics. The same process should have repeated for all six letters.

Figure 5-3. Spelling Out GOTHIC on the Display

The Letters Box

The Letters box contains an ASCII representation of the letters you type. They look like characters printed by a dot-matrix printer, or the way characters normally look when they appear on computer displays. The letters in this box are for reference only; they represent the actual keys you press to display the characters appearing in the Graphics box.

The Grid and Graphics

The screen grid area contains a representation of the characters as defined by the person who created GOTHIC.SET using Animatrix. The Graphics box contains actual-sized images of the characters you type as defined in the character set GOTHIC.SET, and which just appeared giant-sized on the screen grid.

The Cursor

The lit square in the Letters box acts like a cursor; it shows where the next character will appear—not only in the Letters box, but on the grid and in the Graphics box as well. The ← and → keys move the cursor left and right, and the RETURN key moves the cursor down one line and to the left edge.

If you press an arrow key so many times that the cursor goes too far to the left or right, the cursor wraps around to the beginning or end of the next line. If you press an arrow or the RETURN key so many times that the cursor goes past the bottom end of the box, the cursor wraps around to the top again; if the cursor goes past the top, the cursor wraps around to the bottom. Try it now and see what happens. When you are through, move the cursor to the block to the right of the letter C.

Uppercase and Lowercase

On the right side of the display, between the Letters box and the Graphics box, appears the word <UPPER>; this tells you whether Animatrix sees the key you press as an uppercase or a lowercase character. Press CONTROL-Z to change this word to <lower>; later on, you can press CONTROL-A to type more uppercase letters. Press RETURN, and notice that the lit square in the Letters box drops down one line and moves over to the left edge. Then type the following letters one at a time, waiting for the cursor to come back before typing the next letter:

```
compute
```

CONTROL-Z for lowercase

CONTROL-A for uppercase

the CONTROL key is labeled CTRL on the Apple II and II Plus

Ignore for the moment the bar of dots that appears on the grid between the t and the e; you'll have more information about it in a few minutes. Notice, though, that it doesn't appear in the Graphics box. Also ignore the flashing small x (although that's a little like asking you to ignore your tongue), to which we will also return.

Looking at the grid, you can see some interesting differences among the various forms. First is the obvious difference between the uppercase ASCII characters in the Letters box and their Graphic representations in the screen grid. But there's also a difference between a letter in the grid and the same letter in Graphics. Notice how much rounder and smoother the Graphics characters appear.

The Character Set Display

Press CONTROL-D to see the entire Gothic character set at once. This display appears:

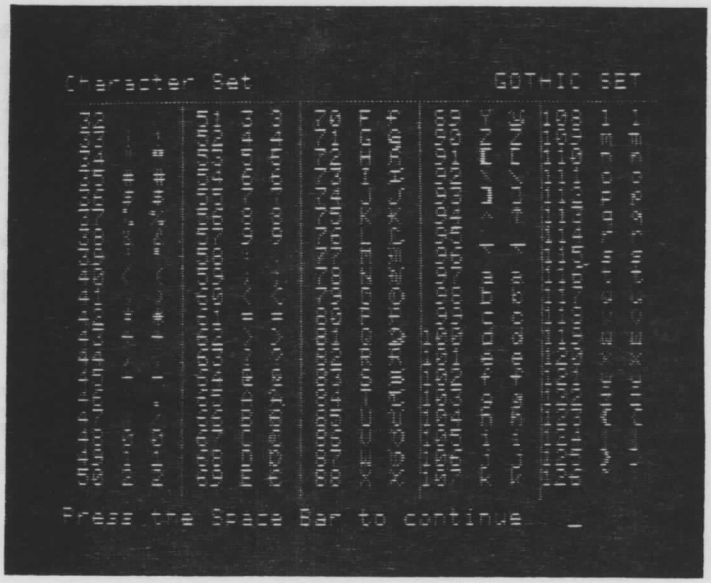


Figure 5-4. The Gothic Character Set

left—ASCII code; middle—ASCII char; right—created char

In each of the five columns, the number at the left represents the ASCII code, the character in the middle is the standard ASCII character, and the character at the right is the newly defined graphic. For instance, the number 65 represents the capital letter A. Look at 65 and notice its associated characters.

By the Way: The numbers begin at 32 because the first 31 numbers aren't available for character assignment. They are all control characters, which are invisible to begin with and are all used by Animatrix's companion program, HRCG, covered in Chapter 6. The blanks after the number 32 aren't really blanks; in the ASCII code, 32 represents the SPACE character.

For more information on ASCII, see the Applesoft reference manual.

Now press the SPACE bar to return to the grid.

The Command List Display

To see the command list, press CONTROL-I. This appears on your display:

```
Hi-Res Character Set Editor Instructions

ctrl-A  :  Shift to UPPER case letters
ctrl-D  :  Display entire character set
ctrl-I  :  Display editor instructions
ctrl-K  :  [  (left square bracket)
ctrl-L  :  \  (backslash)
ctrl-O  :  _  (underscore)
ctrl-S  :  Assign same character image
ctrl-T  :  Toggle high bit (1/2 dot shift)
ctrl-Z  :  Shift to lower case letters
escape  :  Quit and save character set

Paddle  0  moves the cursor horizontally,
Paddle  1  moves the cursor vertically.
Button  0  turns graphic dots (bits) off,
Button  1  turns them on.  The keyboard
controls all other editing features.

To enter letters to be edited, simply
type them on the keyboard, or use the
forward & back arrows and return key.

Press the Space Bar to continue...
```

Figure 5-5. The Command List Display

You've already seen the first three commands. The fourth, fifth, and sixth commands are for Apple II users; the keyboard on the Apple II doesn't have keys for the left bracket ([), backslash (\) and underscore (_) characters so Animatrix provides a way to access them. The same control characters, by the way, are used in APA (see Chapter 1) to get the same results.

The rest of the commands have to do with actually creating and editing characters on the grid. Read over all the commands on the display and then press the SPACE bar; you can go over each of the remaining commands when you have the grid in front of you so that you can practice using them.

Entering and Editing Characters

To edit or create an image for a given character, first decide where you want it to go on the display. Then, using the arrow keys and the [RETURN] key, put the cursor in that position; you can see the cursor move around in the Letters box.

When you are creating and editing fancy letters or non-English alphabets, where you place a particular letter on the display has no bearing on how the letter will look when you use it in your own programs. The positioning of characters, however, matters a great deal when you are creating a block of characters used together to make up an animation graphic. Luckily, you needn't worry about that until you get to the next chapter.

Entering a Character

Move the cursor to the first column in the third row down. Press [CONTROL]-[A] to set uppercase, then type the letter Y. The letter appears in all three boxes in relatively the same position. To recap: when you type a character, its standard alphabetic image appears in the Letters box and its graphics image, if it's defined in the character set, appears in the Graphics box and on the grid.

If you are defining a new character set, the Graphics box and the grid will both be empty, even if you type madly and fill the Letters box. While you have stated with your keypress what key will stand for a graphics character, you haven't yet defined the character the key will produce.

Editing a Character

Once you have chosen and pressed the key that will produce the new (or edited) character, you're ready to begin work. Now's the time to take notice of the small flashing × at the bottom left of the display.

The grid is made up of 35 large rectangles. Each rectangle is made up of a block of dots, 7 columns wide by 8 rows high. Each dot can be filled or empty. You use the little blinking × to determine which dot you're working with.

You can move the × around in the following ways:

- To move horizontally, turn hand control 0
- To move vertically, turn hand control 1

The × marks the spot (sorry) where you make your mark (sorry again) when you press the button on control 1. A dot is erased when the × covers it and you press the button on control 0.

To get some practice, move the cursor to the lower left corner of the grid. Now type the number 7. Next, move the × to that box and put a small bar across the center of the number by filling in the fifth row from the top of the box, columns 2, 3, 5, and 6. Notice as you fill each dot that the image in the Graphics box changes.

Figure 5–6. Editing a Character

As you type a dot, the image is also changed in the character set table. Call up the whole character set by pressing CONTROL-D; if you look at ASCII number 55, the code for 7, you'll see that the changes you made have been duly entered into the character set table.

Moving Half a Dot

You've noticed how much rounder and smoother the created characters appear in the Graphics box than on the grid. One of the reasons is their size: since they're smaller, they appear smoother. And in some cases, the characters actually do have smoother contours, thanks to something called a half-dot shift. Animatrix has the ability, commanded by a CONTROL-T, to shift a row of horizontal high resolution dots one-half dot to the right. Here's a quick demonstration:

What You Type	What You Get
1. Position the × over the bar you just added to the character 7.	
2. Press CONTROL-T while watching the 7 in the Graphics box	You can see the shift occur
3. Press CONTROL-T again	The bar shifts back to the left

Figure 5-7. Moving Half a Dot

Notice that, in the grid, the bar does not shift. Instead, a dot lights up in the border between the block containing the 7 and the next block over to the right so that you can see when a line has been

CONTROL-T for a half-dot shift

shifted. The column of dots to the right of the uppercase T shows that all of the lines of that character have been shifted.

Duplicate Character Changes

In Animatrix, a change in one character's block makes the same change in all blocks of the same character. To see this in action, type the uppercase letter C with the cursor over any blank block (that is, a block with no character drawn in it). Now you'll have two Gothic C's on the display—the one Animatrix just typed, and the one in the word GOTHIC. And now, make a change to one of the C blocks by lighting up a dark dot or by turning off a lit one. Now look at the other C—the change happened there as well.

This feature can produce some unexpected results. Move the × to any blank block. Now turn on any dot in the block. Every blank block in the grid has the same dot turned on. That's because Animatrix sees the blank block as representing ASCII 32—the SPACE character. Press CONTROL-D and look at the first character; what is blank to the ASCII character has a dot in it to the Graphics character.

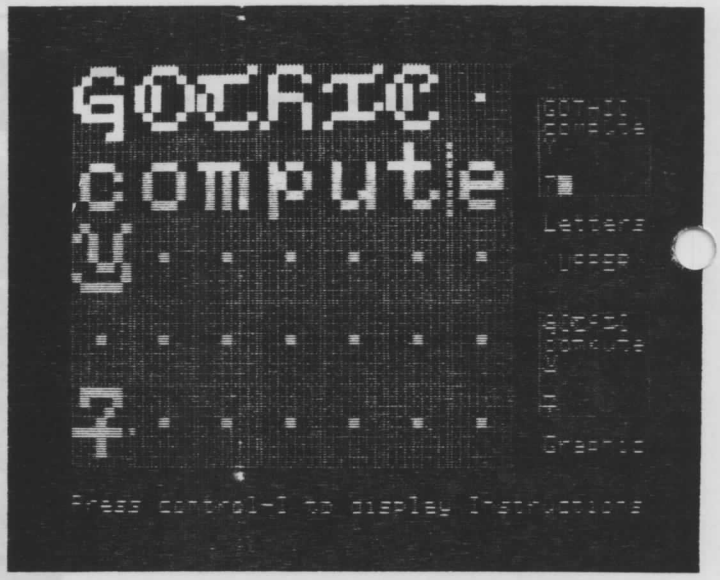


Figure 5-8. Character Set with SPACE Defined as Dot

Creating Duplicate Characters

Any undefined character can be made a duplicate of a defined one so that both characters, although having different names, have the same image. You'd use this feature to create a character similar in many respects to a character you had already created. For instance, look at the lowercase d and the lowercase o in GOTHIC.SET. The designer of this set might have created the d first, then duplicated it under the name o; finally, the designer modified the character until it looked like the o was intended to look.

To make this feature work, pick a defined character and one yet to be defined. Imitating GOTHIC.SET's creator, assume that the defined character is d and that the undefined character is o. Move the cursor to a blank square in the Letters box. Type d, so that its image shows on the grid. Now backspace the cursor so that it's over the d you just typed, and then press (CONTROL)-(S) (for Same). The message

(CONTROL)-(S) duplicates a character

appears on the display; type the letter o.

Under ordinary circumstances, the letter o would replace the letter d in the Letters box and you could happily go about changing the newly created o image to your heart's content. In this case, however, you get the message

You can't give a name to an image if that name has already been used. And, unfortunately for this demonstration, all the characters in GOTHIC.SET have already been used. To experience this command fully, you'd have to leave the program and restart it; then press (RETURN) without typing any characters when the program asks you what character set you want to edit, and begin to define your own character set. As luck would have it (heh heh), you've covered all of Animatrix's features except the various options for leaving the program, so you're about to get your chance.

Leaving Animatrix

To exit the program, press ESC. Since you've been editing an existing character set, Animatrix asks if you want to save the character set, presumably changed by your edits, under its old filename. If you affirm that you do, the character set you have edited replaces the original one on the disk. If you say that you don't, or if you've been creating a new character set, Animatrix asks for a new filename. When you type the new filename and press RETURN, the new character set is saved under the new name and the old character set (if there is one) is undisturbed. If you press RETURN without typing a filename, the new character set is erased as you leave the program.

▲ **Warning**

If your only copy of DOS Programmer's Tool Kit Volume I is the one in the drive (shame on you!), just press RETURN; otherwise, you'll lose the original version of GOTHIC.SET. You might want to use it in its unaltered form some day.

last chance to back out

At this point, you still have a chance to back out; Animatrix asks you if you really want to leave the program. If you say that you do, then all that was just described happens. If you say that you don't want to quit, you return to the program with your current character set still in memory and your old character sets unchanged.

When you leave Animatrix, the High Resolution Character Generator (HRCG) functions are still active; Animatrix uses HRCG to perform its wonders. You'll find HRCG explained in detail in the next chapter.

Before You Go...

Here are a few exercises for you, both for the fun of it and to demonstrate certain principles associated with HRCG and Animatrix. You might not totally understand how some of what you're about to see happens; it will make more sense after you've read the next chapter.

When you told Animatrix you wanted to work with GOTHIC.SET, that character set was loaded into memory. In fact, it's still there, even after you leave Animatrix. Assuming that you're back to the Applesoft prompt, press the following characters:

CONTROL-A 1 CONTROL-X

Now type the LIST command and press RETURN.

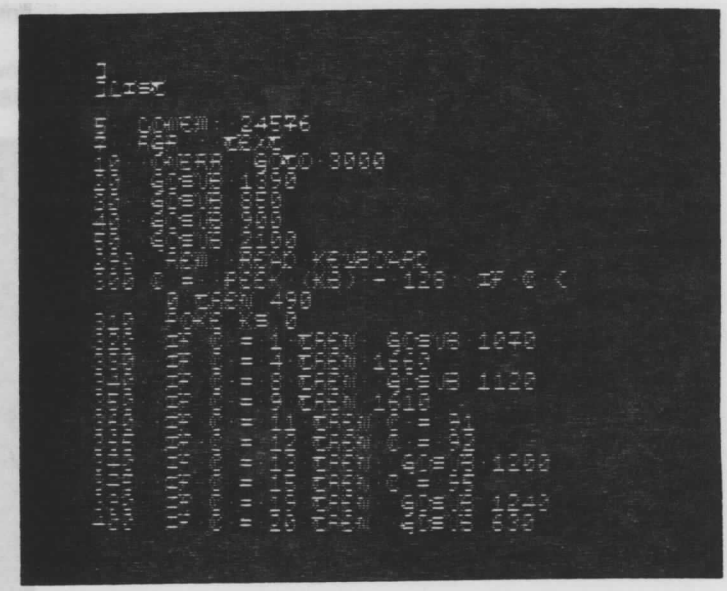


Figure 5-9. Listing in Gothic

As noted earlier, Animatrix makes use of HRCG. Even though you've left Animatrix, HRCG and the GOTHIC.SET character set are both still in memory. CONTROL-A is one of HRCG's commands; it lets you decide which character set will be active. The number 1 you just typed tells the computer to use character set number 1—which in this case happens to be GOTHIC.SET. The CONTROL-X at the end of the sequence prevents the Apple IIe's error bell from ringing and stops an error message from appearing on the display; the computer thinks CONTROL-A 1 is an error (which shows you how dumb computers really are).

This second exercise lets you see how to use Animatrix to create characters for animation. Type FP to reset HIMEM:, and rerun Animatrix. Load the character set FROGS1.SET and type in the following array of letters to the grid, exactly as they appear:

①Last character is a space

②First two characters are spaces
③Two spaces between # and A
④Last five characters lowercase

```
①ABCDEQ
 FGHIJKR
②  LMNOP
③!#  ABC
④$%lmnoj
```

These graphics, used in the RIBBIT program, give good examples of how the various display segments look when you are doing

character design. HRCG will cover the details of the process—like how to get the array defined above to appear the way it does in Animatrix's grid.

Now for the third and final exercise. Leave the program, load RIBBIT, and list line 10400. You should see this on your display:

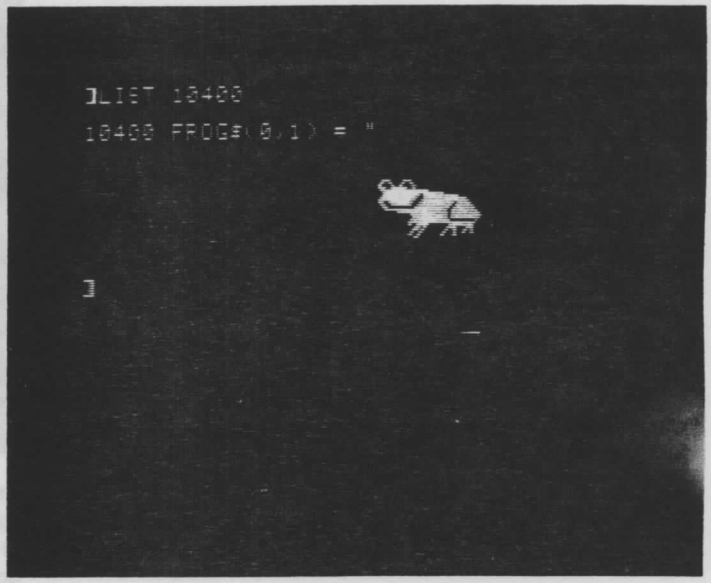


Figure 5-10. RIBBIT, Line 10400

Look familiar? It's part of the image that appeared on Animatrix's grid. Now press CONTROL-RESET, turning off the high resolution displays and revealing the *real* characters that make up the array forming the frog:

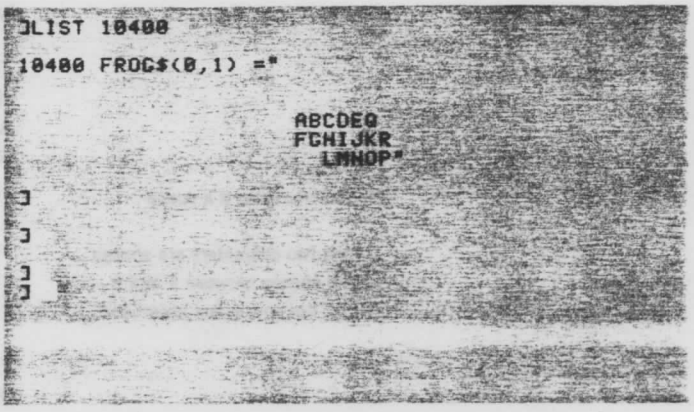


Figure 5-11. RIBBIT, Line 10400, with High Resolution Displays Off

Finally, list line 10400 again—and see how it looks in plain old BASIC code—with a bunch of invisible embedded control characters:

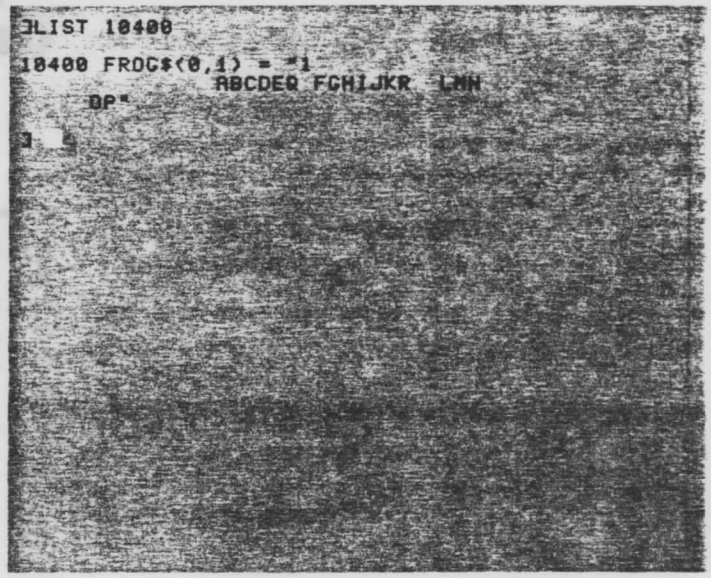


Figure 5-12. RIBBIT, Line 10400, in BASIC Code

The answers to the dozens of questions we just *know* are filling your amazed and bewildered mind are to be found in the very next chapter, High Resolution Character Generator.

Practice Makes Perfect

You covered a lot of ground in this chapter. Before going on, you might want to get in some practice creating some character sets. The real learning with Animatrix is in the doing!

High Resolution Character Generator

High Resolution Character Generator

High Resolution Character Generator (HRCG) is a set of assembly-language routines for showing text on the high-resolution graphics displays. HRCG provides an extensive set of CONTROL key commands that replace and augment the usual text display commands. You can use these commands to

- mix onscreen text and graphics
- combine images from both page 1 and page 2 of high resolution graphics
- create animation sequences

HRCG is a companion program to Animatrix (Chapter 5), and was used with that program and the Relocating Loader (Chapter 7) to create MAXWELL, RIBBIT, and SKYLAB. You'll find all of these programs on the disk labeled DOS Programmer's Tool Kit Volume I.

HRCG was designed to be used with other programs; Chapter 8 demonstrates how you can do this, and provides some examples of how to use HRCG for animation.

How This Chapter Is Organized

This chapter has two sections. Section 1 is a tutorial and should be read while you're sitting at your computer. Following along and trying out the different examples will give you a real sense of what HRCG is all about. Section 2 is a reference section. It contains summary tables of all commands and options, and other important technical data.

Warning

HRCG won't work correctly with the 80-Column Text Card active. Random characters may appear on the display, the text window might inexplicably be set to a width of one column, and other unpredictable events might occur. If you are using an Apple IIe with an 80-column card, be sure to turn the card off before starting HRCG.

Section 1

HRCG Tutorial

Getting HRCG Up and Running

HRCG was written in relocatable format (see the *6502 Assembler/DOS Tool Kit Manual*); it uses the Relocating Loader to install itself in the proper place in memory. To get it running, type:

```
FP

RUN LOADHRCG
```

Technolalia: FP clears the computer's memory and resets HIMEM:. LOADHRCG sets the Relocating Loader into motion. The Loader places the Rfile called HRCG at the highest point available for program and variable space (HIMEM:); then it resets HIMEM: to just below HRCG so that HRCG can be in memory at the same time as the Applesoft program you are writing or running.

See Chapter 7 of this manual for a more detailed description of the Relocating Loader. See the *6502 Assembler/DOS Tool Kit Manual* for a description of Rfiles.

HRCG asks you how many character sets you want to load from disk. You can select up to nine character sets. Then HRCG asks you for the name of each character set, one at a time, and loads each character set you name (assuming the set you name is on the disk, of course). For purposes of demonstration, you'll be using four character sets—GOTHIC.SET, CYRILLIC.SET, GREEK.SET, and FROGS1.SET. If you're following along with your computer, tell HRCG to load these four sets.

ASCII is the standard character set

If you tell HRCG that you have no character sets to load (that is, if you type 0), HRCG still in fact uses one set—the standard ASCII set. Just so you'll know, the ASCII set is not generated by your computer's built-in hardware character generator when you use HRCG, but is in fact a software character set built into HRCG.

Warning

Each named character set must be on the disk in the most recently used drive. If you name a character set HRCG can't find, the program stops with an UNABLE TO LOAD message.

Also note that the names of all of the character sets that came with this package end in the suffix .SET. While this suffix was arbitrarily chosen (and you need not use it with any character sets you create yourself), it is part of the names of the sets that appear on the disk. Thus to load the GOTHIC.SET, it isn't enough simply to type GOTHIC; you must type its full name, GOTHIC.SET, before you press (RETURN).

The drives click, hum, and load each character set as you name it. Finally HRCG clears the display, and this appears:

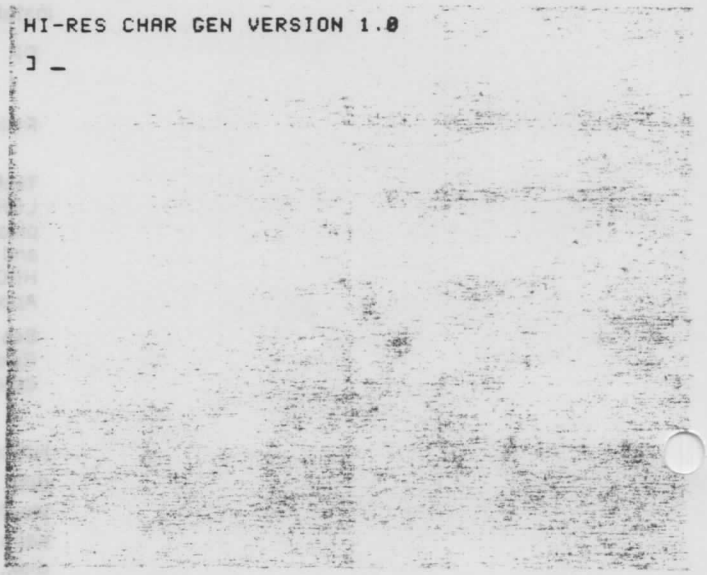


Figure 6-1. HRCG Opening Display

While the display you see before you is modest in appearance, it does tell you that HRCG is up and running. The blinking underline cursor next to the Applesoft prompt is HRCG's signature.

Alternate Character Sets

Now list line 100: the line looks like plain old Applesoft with the plain old character set. But you're actually looking at a special software character set loaded from the disk.

So that you can see that there really are other character sets available, type in this example:

What You Type	What Happens
1. Press CONTROL-A	Tells HRCG to use the character set that follows
2. Type 1	Names character set you designated as 1—the GOTHIC.SET
3. Press CONTROL-X	Intercepts and prevents Applesoft's SYNTAX ERROR message (Applesoft doesn't recognize CONTROL-A 1 as a command)
4. Type LIST 100 and press RETURN	Standard Applesoft LIST command

the CONTROL key is labeled CTRL on the Apple II and II Plus

Note that you only need to press RETURN once, at the end of the line. Usually to activate HRCG commands you needn't press RETURN at all; you have to press RETURN here because of the standard Applesoft LIST command. Here's what your display should look like:

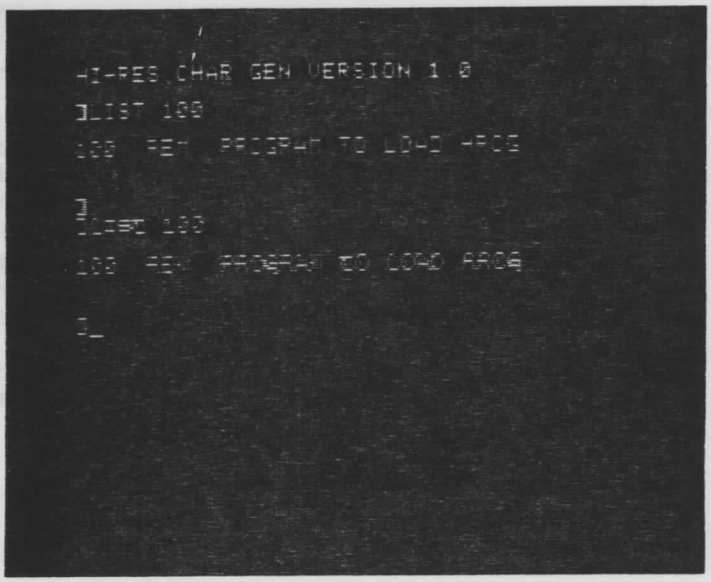


Figure 6-2. BASIC Program Line in Gothic Characters

[CONTROL]-[A] <n> designates character set

What appears on your display is a BASIC program line written in GOTHIC.SET characters, the set you designated as #1 back when you loaded HRCG. [CONTROL]-[A] is your first HRCG command, it says "use the following character set." The number that you then type tells the machine which character set to use.

About [CONTROL]-[X]: Applesoft gets confused by certain [CONTROL] sequences, and assumes them to be syntax errors. Pressing [CONTROL]-[X] makes Applesoft think that you've cancelled the confusing [CONTROL] message, which HRCG has already received and processed (chuckle).

Use the [CONTROL]-[X] option with [CONTROL]-[A] only for immediate execution. When you use HRCG command codes in actual program lines, don't use [CONTROL]-[X].

Try the following lines of code, typing them one at a time and remembering to press RETURN at the end of each line:

What You Type	What You Get
1. [CONTROL]-[A] 2 [CONTROL]-[X] LIST 100	Lists a line in CYRILLIC.SET
2. [CONTROL]-[A] 3 [CONTROL]-[X] LIST 100	Lists a line in GREEK.SET
3. [CONTROL]-[A] 4 [CONTROL]-[X] LIST 100	Lists a line in FROGS1.SET
4. [CONTROL]-[A] 0 [CONTROL]-[X] LIST 100	Lists a line in standard ASCII set

Now go through the sets again, this time listing the whole program in memory for each of the five sets (set 0 counts). Notice that when you list a program in HRCG, the display scrolls in a strange way; it seems to move in waves, rather than in the smooth way you are used to. What you are seeing is a high-resolution display scroll, rather than a text display scroll. Don't be concerned about the technicalities; use the information to recognize when HRCG is in effect.

For Your Information: All of these sets were designed using Animatrix. Later, you can run Animatrix and examine them in detail.

Manipulating the Display

Here are the commands you can use in HRCG to manipulate characteristics of your text display. Commands with the asterisk (*) are HRCG's standard features—that is, when you run LOADHRCG these commands are automatically in force.

Table 6-1. Display Commands in HRCG

Command	Effect	Comment
CONTROL-I	Inverse video	Replaces Applesoft INVERSE command
CONTROL-N	*Normal video	Replaces Applesoft NORMAL command
CONTROL-K	*All uppercase	K for Kaps Lock (spelling doesn't kount)
CONTROL-L	All lowercase	L for Lowercase
CONTROL-S	Shift	Next character upper, all after lower

FLASH isn't available in HRCG.

Type the following into the computer (ignore the error message that you'll get):

What You Type	What Happens
1. Press CONTROL-A	Uses the following character set
2. Type 0	Chooses set 0 (ASCII.SET)
3. Press CONTROL-S	Displays next letter typed in uppercase
4. Type THIS IS	
5. Press CONTROL-I	Sets highlighted text
6. Type REALLY	
7. Press CONTROL-N	Restores normal text
8. Type A	
9. Press CONTROL-A	Uses the following character set
10. Type 1	The number for GOTHIC.SET
11. Press CONTROL-K	Uses uppercase until canceled
12. Type STRANGE	

What You Type	What Happens
13. Press CONTROL-A	Uses the following character set
14. Type 0	Back to standard ASCII.SET
15. Press CONTROL-L	Back to lowercase
16. Type LINE.	

This appears on your display:

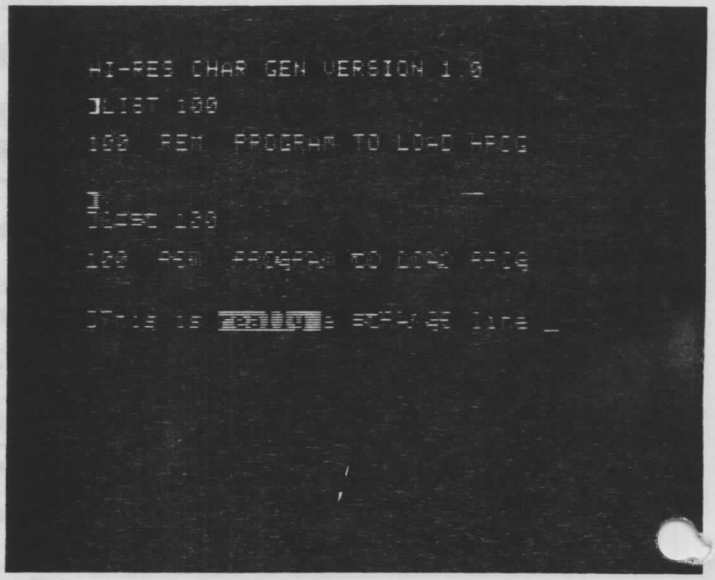


Figure 6-3. Sample Using HRCG's Display Commands

Clearing the Screen

Here are some commands to clear part or all of the screen. These commands replace the ones used with the Apple's normal text display; HRCG works on the high-resolution graphics displays instead. The Applesoft commands they replace or simulate are listed to the right.

Table 6-2. Screen-Clearing Commands in HRCG

Command	Effect	What It Replaces
CONTROL-P	Clears page	ESC @ or CALL -936
CONTROL-E	Clears to end of line	ESC E or CALL -868
CONTROL-F	Clears to end of page	ESC F or CALL -958

CONTROL-Q homes cursor without clearing display

CONTROL-Q, HOME, and CALL -936 cause the cursor to move to the upper left corner of the display, but don't clear any text. The other CALLs don't work at all. The ESC codes function as they usually do.

Display Pages and Fancy Changes

In order to demonstrate the rest of HRCG's features, you need to run another program. Make sure that the DOS Programmer's Tool Kit Volume I disk is in the active drive. Next press CONTROL-RESET and type FP to clear memory and to reset HIMEM:. Finally, run MAXWELL. After the show, come back here; you'll know it's over when the flashing underscore appears.

MAXWELL was created using Animatrix and the Relocating Loader, and it uses HRCG extensively. When you run MAXWELL, HRCG is automatically loaded and activated along with MAXWELL's code.

Choosing the Display Page

Your computer has two pages of high-resolution graphics display. Programs like MAXWELL use both displays. What you have on your display now is a combination of material from both **page 1** and **page 2**, all of it displayed on page 1 (what did he say?). A demonstration will make things clearer. First, here are the commands to choose and to display each of the two high-resolution graphics pages:

Table 6-3. Graphics Page Commands in HRGC

Command	Effect
CONTROL-O CONTROL-A	Page 1 as primary page
CONTROL-O CONTROL-B	Page 2 as primary page
CONTROL-O CONTROL-D	Displays current primary page

A Note About CONTROL-O: A number of CONTROL key commands in HRCG are preceded by CONTROL-O (for Options). Using a combination like this expands the number of possible commands available to you.

The **primary page** is that page to which any text or graphics will be written; the **display page** is that page currently showing on the display. As you'll see in a moment, it is possible to write to one page while displaying another.

The page you're looking at now is page 1, which also happens to be the primary page. To see page 2, press [CONTROL]-[O] [CONTROL]-[B] [CONTROL]-[O] [CONTROL]-[D]. Remember—you don't need to press [RETURN].

Only the high-resolution apple is here: both Maxwell and the text seem to have disappeared. Actually, you've just switched to page 2. Now you'll switch back, but this time in stages. Watch the cursor and just press [CONTROL]-[O] [CONTROL]-[A].

Note that the cursor has disappeared. Actually, the cursor is at the bottom of page 1. It's doing its job, pointing to the next display position where a character typed from the keyboard will appear. While page 2 is the *display* page, page 1 is the *primary* page. To get back to page 1, and to see the cursor again, press [CONTROL]-[O] [CONTROL]-[D].

While you're here on page 1, press [RETURN] to scroll the display a line or two. Then swap over to page 2 via [CONTROL]-[O] [CONTROL]-[B] [CONTROL]-[O] [CONTROL]-[D]. As you can see, the images of the high-resolution apple are no longer matched with each other. Earlier you looked at a display composed of material from both page 1 and page 2, displayed all on page 1. Here you see page 1 and page 2 as two separate entities.

You can never see page 1 and page 2 at the same time; they are located in different places in memory. But what you *can* do, with HRCG's help, is to display one page, keep its image on the display, and write new material on top of it—which you'll soon see.

Page Scrolling

List through line 100 of the program in memory (type `LIST -100`). The program lists in the way you'd expect it to (albeit with a bit of a wave); the text scrolls up the display, with the top line disappearing when the display is full and a new line coming up from the display bottom. Normal scrolling is HRCG's standard condition. But you can also turn off normal scrolling, and substitute a wraparound effect instead.

Here's an exercise to see wraparound. First, press [CONTROL]-[O] [CONTROL]-[P]. This command restores certain standard parameters that make it easier to demonstrate HRCG's features; you'll get more details later (trust us on this one).

Now press CONTROL-O CONTROL-W CONTROL-X (the CONTROL-X is to avoid an error message). Notice that the cursor is up at the top left of the display in the home position. Now list through line 500, and watch the display.

use CONTROL-P to clear display

If that was too fast for you or if the display became too jumbled for you to see what happened, clear the display with CONTROL-P, set `SPEED= 100`, and type `LIST -500` again.

As you can see, when text reaches the bottom line of the display it wraps back up to the top. New characters replace old characters that are in the same display position; but old characters in display positions not needed by the new text remain where they are. This effect is called Wrap Text in the rest of this chapter; the scrolling effect you're used to in regular Applesoft is called (naturally enough) Scroll Text.

Type `SPEED= 255` and clear the display with CONTROL-P in preparation for experimenting with **overlays**.

Overlays

Overlays occur when characters on a page are combined with other characters—that is, when a second character is *laid over* one that is already on display. These next few commands work differently with Wrap Text in effect than they do with Scroll Text active. Try them in different combinations; you'll get some remarkable results!

Display Clearing in Wrap Text: When any overlay command is in effect at the same time as is Wrap Text, the display-clearing action of CONTROL-P disappears. To clear the display, press CONTROL-O CONTROL-P CONTROL-P and follow it with the particular overlay command you want to use. CONTROL-O CONTROL-P turns off any overlay; CONTROL-P then clears the display. If you use Wrap Text, it's a good idea to clear the display every so often; if you don't, the display is going to get messier than a shirt in a soapsuds commercial.

The sets you are using are all high-resolution graphics characters, each made up of a series of dots. When the descriptions that follow speak of dots, they're refering to the basic elements of each character.

Here's a list of the overlay commands; following the list you'll find detailed explanations of each command.

Table 6-4. Overlay Commands in HRCG

Command	Name	Effect
CONTROL-O CONTROL-O	Text Overstrike	Old dots combine with new dots on same page
CONTROL-O CONTROL-C	Complement Overlay	New dots covering old dots on same page cause color of old dots to be complemented
CONTROL-O CONTROL-T	Transparent Overlay	New dots overlay character on secondary display; result transferred to primary
CONTROL-O CONTROL-R	Reverse Overlay	Same as Transparent, except overlaid dots on secondary display complemented
CONTROL-O CONTROL-P	Print Replace	The standard condition

Text Overstrike

Text Overstrike: CONTROL-O CONTROL-O

With Text Overstrike in effect, when a new character attempts to occupy the same space as an old character all the dots of the old character remain in place; the new character just gets displayed on top of the old one.

Here's a demonstration of Text Overstrike:

What You Type	What Happens
1. Press CONTROL-O CONTROL-O	Sets Text Overstrike
2. Type T	
3. Press CONTROL-H	Another way of backspacing—you can also use ←
4. Type the plus sign (+)	You end up with a crossed T

Try Z CONTROL-H N CONTROL-H T to get a crosshatch.

Complement Overlay

Complement Overlay: CONTROL-O CONTROL-C

Complement Overlay, summoned via CONTROL-O CONTROL-C, causes new dots covering old dots to display their complementary color. These are complementary colors:

Black . . White
Blue . . . Orange
Green . . Violet

In this example, the color white will be complemented by the color black. Set Complement Overlay now and, using the same text you used for the last example, type T and press CONTROL-H + (no spaces—they appear here for clarity). Instead of adding a cross to the T, the + causes part of the T to disappear. If you press CONTROL-H and type + again immediately, the vanished section of T reappears.

A More Formal Demonstration: There's an example called FLASH at the end of the chapter using the principles just demonstrated, but in program form. This program will do automatically what you just did manually—but with slightly flashier results. Don't run it now, though; you'll wipe out MAXWELL, which you'll need in memory if you want to follow the rest of the tutorial.

Transparent and Reverse Overlay

Transparent Overlay and Reverse Overlay take advantage of the high-resolution graphics displays by combining elements from both pages. With each of these commands, material from the secondary page is combined with material from the primary page; then the combined material is transferred to the primary page and is displayed.

Transparent Overlay: CONTROL-O CONTROL-T

With Transparent Overlay (accessed by pressing CONTROL-O CONTROL-T), a character on the primary page is laid on top of a character from the secondary page. It works in much the same way as Text Overstrike, except that images from two pages are combined.

Reverse Overlay: CONTROL-O CONTROL-R

Reverse Overlay (CONTROL-O CONTROL-R) works slightly differently, in that instead of just combining the images, any dots in the character on the secondary page are complemented before the transfer to the primary page takes place. It's like Complement Overlay using two displays.

The best way to understand the differences between these commands is to experiment freely. For your edification, here's directed experiment:

What You Type	What Happens
1. Press CONTROL-O CONTROL-B	Names page 2 as primary...
2. Press CONTROL-O CONTROL-D	...and displays it
3. Type `IN POETS AS TRUE GENIUS IS BUT RARE,`	
4. Press ← to start of line	
5. Press CONTROL-O CONTROL-A	Names page 1 as primary...
6. Press CONTROL-O CONTROL-D	...and switches to it
7. Press CONTROL-O CONTROL-T	Sets Transparent Overlay
8. Type `TRUE TASTE AS SELDOM IS THE CRITIC'S SHARE`	
9. Press CONTROL-X	Avoids error message

After observing the results, repeat the process, substituting Reverse Overlay (CONTROL-O CONTROL-R) for Transparent Overlay. You'll end up with something like this on your display:

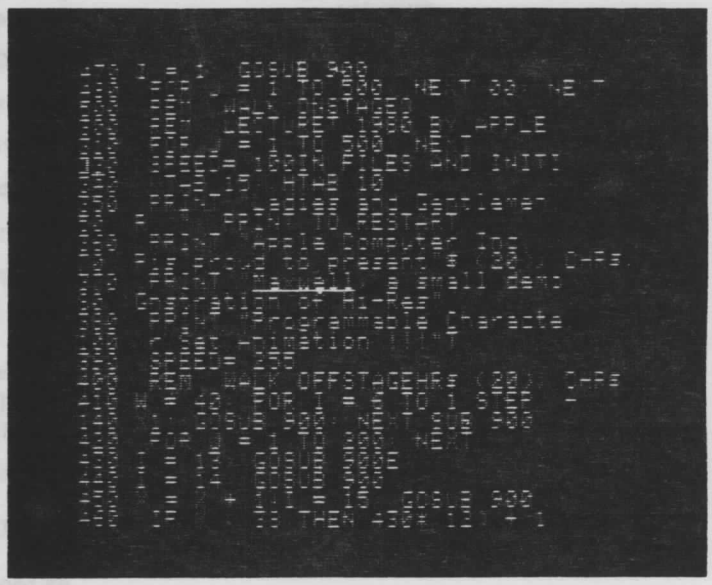


Figure 6-4. Sample of Reverse Overlay

Print Replace

Print Replace: CONTROL-O CONTROL-P

The final command in the overlay series is actually the overlay cancellation command, Print Replace. You activate it with CONTROL-O CONTROL-P; thereafter a new character laid atop an old one replaces the old character. Print Replace is the standard condition of HRCG.

Windows

As with regular text-display Applesoft, you can set text windows in HRCG. There are three commands to set windows:

Table 6-5. Window-Setting Commands in HRCG

Command	Effect
CONTROL-Y	Full-screen window
CONTROL-V	Upper left corner window
CONTROL-W	Lower right corner window

CONTROL-V combines Top of Window and Left Edge in one command; CONTROL-W likewise combines Bottom of Window

CONTROL-Z restores standard parameters

and Right Edge. To see these commands in operation, begin wi[illegible] all parameters reset to their original values. Luckily, HRCG provides the command CONTROL-Z, which restores all standard features. The standard features are:

- ASCII.SET
- Print Replace
- Normal video
- Uppercase
- Scroll Text
- Page 1 primary
- Text window full (no windows set—standard full-screen display)

After restoring the standard parameters, be sure the display is clear (CONTROL-P CONTROL-X) and type this line of direct execution code:

```
VTAB 10: HTAB 12: PRINT CHR$(22)
```

Note: CHR$(22) is CONTROL-V

As soon as you press RETURN, CONTROL-V is activated at the new position. Now type this line to set the lower right corner, and then list the program:

```
VTAB 20: HTAB 20: PRINT CHR$(23)
```

Note: CHR$(23) is CONTROL-W

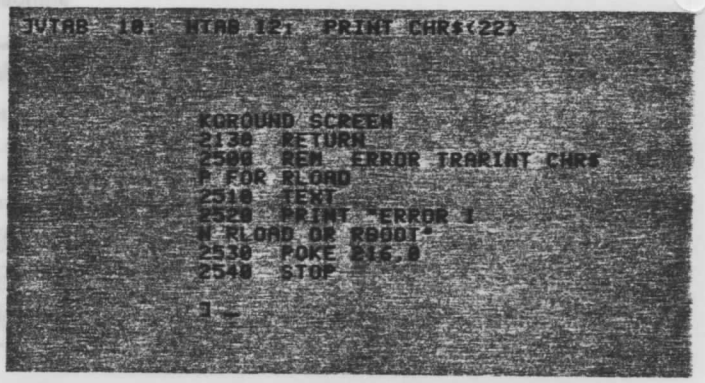


Figure 6-5. Sample of Lower Right Corner Window

To restore full display, use CONTROL-Y or CHR$(25).

Block Display—Soul of Character Animation

In listing the whole MAXWELL program, you might have noticed a small person wandering between lines 1010 and 1180. If you didn't notice, then list those lines now. That little guy is Maxwell himself; what you're seeing is a block of characters defined in Animatrix and given life by HRCG. Maxwell is actually made up of a collection of individual characters, held together in Maxwell's form by HRCG's Block commands.

HRCG has two Block commands, one for marking the start of a group of characters to be taken as a whole, and one for marking the end of the group:

Table 6-6. Block Commands in HRCG

Command	Name	Effect
CONTROL-B	Start Block Display	All characters after this command taken as single unit until CONTROL-D occurs
CONTROL-D	End Block Display	Defines end of block; each character typed after this taken as individual unit again

Line 1020 of the MAXWELL program uses the Block commands. List line 1020 by itself:

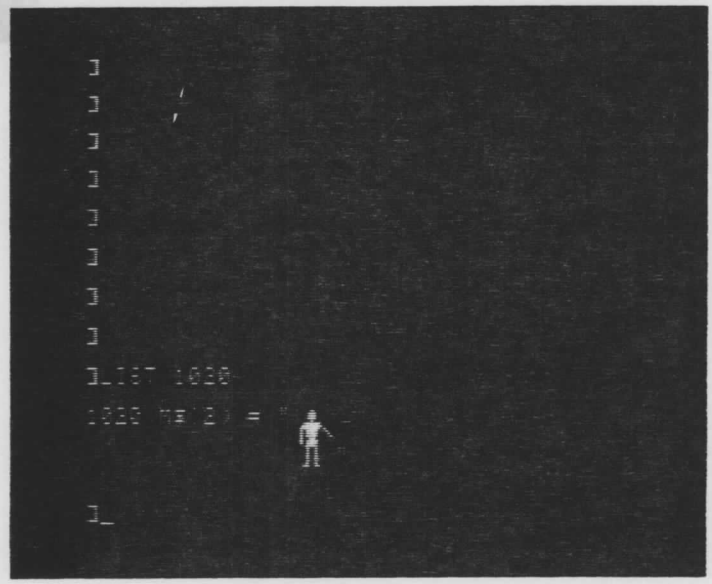


Figure 6-6. Maxwell Viewed Through HRCG

The figure of Maxwell is actually made up of eight special graphics characters taken as a block. The characters are from MAX.SE and are a grouped combination of the characters named D, A, G, E, B, R, F, and C. To see the way they're combined, press CONTROL-RESET:

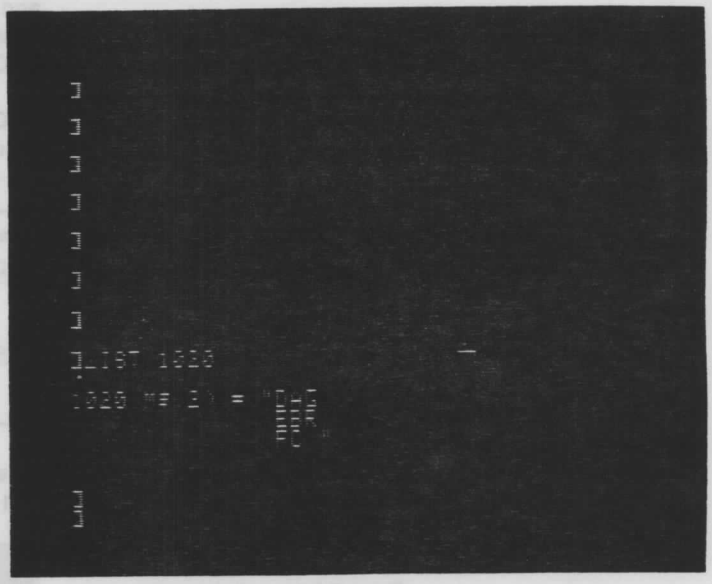


Figure 6-7 Maxwell in Normal Text Display

Pressing CONTROL-RESET dumps HRCG and restores the normal output of the text display. The way the characters are grouped might well remind you of how the Letters box in Animatrix looks with some characters typed in. As it turns out, that's exactly what it is. Later on, you can type FF and run ANIMATRIX to prove this for yourself. But now list line 1020 again:

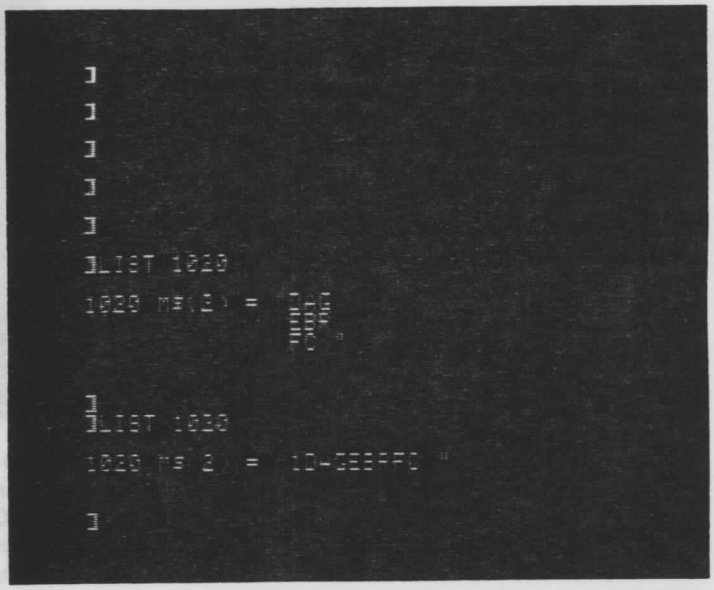
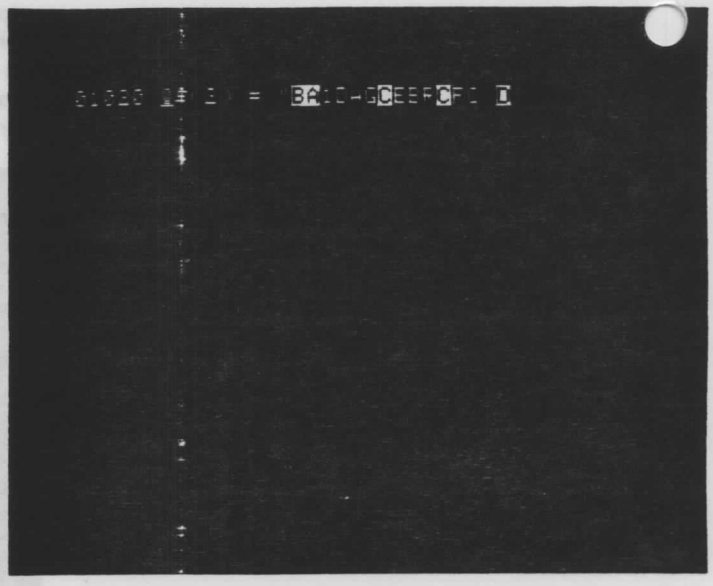


Figure 6-8. Listing of Line 1020

There are embedded control characters in this line to which HRCG is responding . You could execute either BOSTON WINDOW (Chapter 2) or APA (Chapter 1), and by issuing the proper commands see the embedded characters. To save you the trouble:

Figure 6-9. Embedded Characters in Line 1020

Control characters highlighted



The first and last characters in line 1020 are the control characters B and D, which respectively define Block Start and Block End in HRCG. CONTROL-A chooses a character set to use, in this case number 1; in MAXWELL, this is the number of MAX.SET. The remaining letters are the names of the graphics characters that form Maxwell's image.

CONTROL-C issues carriage return/line feed

CONTROL-C is a new command; it issues a carriage return and line feed in HRCG, as if you had pressed RETURN. But in Block Display, the left edge of the display is defined by the position of the first character in the block. It's as if the Block command set up a small text window. The net result is that the grid image produced in Animatrix is duplicated exactly through HRCG.

Line 1020 actually says:

1. Set up Block Display.
2. Use MAX.SET as the character set.
3. Display the graphics characters represented by D, A, and G.
4. Issue a carriage return/line feed (position of D is left edge).
5. Display E, B, and R.
6. Do another carriage return and line feed.
7. Display F, C, and a space.
8. End the Block Display.

If you're confused, type FP and run ANIMATRIX with MAX.SET as the set to be edited. Then type the character names D, A, G, E, B, R, F, and C in the form indicated a few paragraphs ago. What you see will clear up your confusion:

Figure 6-10. Different Ways Characters Can Appear

See Chapter 8 for more information and further demonstrations of the Block commands.

Accessing Machine Language Routines

For the technically minded programmer, HRCG provides two commands for summoning user-supplied machine language routines. They are:

Table 6-7. Commands for Calling Machine Language Routines in HRCG

Command	Effect
CONTROL-O CONTROL-Y	Calls user routine 1
CONTROL-O CONTROL-Z	Calls user routine 2

Issuing the first of these commands does a JSR (machine langu for GOSUB) to relocatable location 10 in HRCG (or location 13 in the case of the second command). There HRCG expects to find your subroutine's address in locations 10 and 11 (or 13 and 14 for Routine 2), low byte first. See the part called Entry Points and Technical Miscellanea in the reference section at the end of this chapter for related information.

Clearing HRCG

There is no elegant way to clear HRCG because of what it does to memory. The only way out is to press CONTROL-RESET, and then to type FP. These commands restore normal display, clear memory, and put HIMEM: back where it belongs.

Warning
Don't use TEXT to leave HRCG. It will get you back to the TEXT display, but HRCG will still be running, and you won't have a cursor. Memory will also be disturbed, and there's no telling what will happen.

FLASH: Here's that example of Complement Overlay you were promised earlier. It isn't earthshaking in its proportions, but it does demonstrate how you can use Complement Overlay to some advantage:

What Code You Type	What You Get
1. `NEW`	Clears out memory
2. `10 PRINT CHR$(15); CHR$(3)`	15 is CONTROL-0, 3 is CONTROL-C
3. `20 PRINT "T"; CHR$(8);`	CHR$(8) is CONTROL-H, the backspace command
4. `30 FOR LOOP = 1 TO 10`	Sets up a loop to run 10 times
5. `40 PRINT "+"; CHR$(8);`	Prints + over the T
6. `50 FOR STALL = 1 TO 200: NEXT STALL`	Slows down the process
7. `60 NEXT LOOP`	Back to line 30

Section 2

HRCG Reference

In this section you'll find summaries of HRCG's commands both by function and in alphabetical order. Additionally, this part of the chapter contains technical information for advanced programmers.

Command Summaries

Functional Command Summary

Display Characteristics

Inverse video	CONTROL-I	replaces Applesoft INVERSE command
Normal video	CONTROL-N	replaces Applesoft NORMAL command
All uppercase	CONTROL-K	K for Kaps Lock
All lowercase	CONTROL-L	
Shift	CONTROL-S	Next character upper, all after lower
Choose character set	CONTROL-A <n>	

Clearing the Screen

Clear page	CONTROL-P	
Clear to end of line	CONTROL-E	
Clear to end of page	CONTROL-F	

Choosing Display Page

Page 1 as primary page	CONTROL-O	CONTROL-A
Page 2 as primary page	CONTROL-O	CONTROL-B
Display primary page	CONTROL-O	CONTROL-D

Page Scrolling

Scroll Text window	CONTROL-O	CONTROL-S
Wrap Text window	CONTROL-O	CONTROL-W

Overlays

Text Overstrike	CONTROL-O CONTROL-O	Old dots combine with new dots on same page
Complement Overlay	CONTROL-O CONTROL-C	New dots covering old dots on same page cause color of old dots to be complemented
Transparent Overlay	CONTROL-O CONTROL-T	New dots overlay character on secondary display; result transferred to primary
Reverse Overlay	CONTROL-O CONTROL-R	Same as Transparent, except overlaid dots on secondary display complemented
Print Replace	CONTROL-O CONTROL-P	The standard condition

Windows

Full-screen window	CONTROL-Y
Upper left corner window	CONTROL-V
Lower right corner window	CONTROL-W

Block Display

Start Block Display	CONTROL-B	All characters after this command taken as single unit until CONTROL-D occurs
End Block Display	CONTROL-D	defines end of block; each character typed after this taken as individual unit again

Accessing Machine Language Routines

Call user routine 1	CONTROL-O	CONTROL-Y
Call user routine 2	CONTROL-O	CONTROL-Z

Issuing the first of these commands does a JSR to relocatable location 10 in HRCG (or location 13 in the case of the second command). There HRCG expects to find your subroutine's address in locations 10 and 11 (or 13 and 14 for routine 2), low byte first.

Original Parameters

Restore parameters — CONTROL-Z

Pressing CONTROL-Z sets the following:

- ASCII.SET
- Print Replace
- Normal video
- Uppercase
- Scroll Text
- Page 1 primary
- Text window full

Clearing HRCG

Turn off HRCG — CONTROL-RESET FP

Alphabetic Command Summary

Default parameters are marked with *.

CONTROL-A <n>	Selects character set n
CONTROL-B	Begins Block Display
CONTROL-C	Carriage return
CONTROL-D	Delimits Block Display
CONTROL-E	Clears to end of line
CONTROL-F	Clears to end of screen
CONTROL-I	Inverse video
CONTROL-K	*Caps lock
CONTROL-L	Lowercase
CONTROL-N	*Normal video
CONTROL-O	Selects option (see list below)
CONTROL-P	Clears page
CONTROL-Q	Homes cursor
CONTROL-S	Shift
CONTROL-V	Sets text window (upper-left)
CONTROL-W	Sets text window (lower-right)
CONTROL-Y	*Sets text window (full screen)
CONTROL-Z	Restores default parameters

CONTROL-O CONTROL-A	*Primary = page 1
CONTROL-O CONTROL-B	Primary = page 2
CONTROL-O CONTROL-C	Complement Overlay
CONTROL-O CONTROL-D	Display primary
CONTROL-O CONTROL-O	Text Overstrike
CONTROL-O CONTROL-P	*Print Replace
CONTROL-O CONTROL-R	Reverse Overlay
CONTROL-O CONTROL-S	*Scroll Text
CONTROL-O CONTROL-T	Transparent Overlay
CONTROL-O CONTROL-W	Wrap Text
CONTROL-O CONTROL-Y	Call user routine 1
CONTROL-O CONTROL-Z	Call user routine 2

Block Character Forms

Below are the four ways you might see the same characters appear on the display. The first figure is seen through HRCG, the second in HRCG after pressing CONTROL-RESET, the third in a normal line listing, and the fourth in a line listing with control characters visible.

Figure 6-11. Different Ways Characters Can Appear

Complementary Colors

Black . . White
Blue . . . Orange
Green . . Violet

Differences from Standard Text Display

Although HRCG is intended to mirror the operation of the standard text display, there are several differences:

- HRCG uses uppercase as its standard. Apple IIe users will notice this difference; Apple II and II Plus users won't.
- The cursor is a flashing underscore.
- `HOME` and `CALL -936` set the print position to the upper left corner of the text window, but they don't clear the display. Use (CONTROL)-(P) instead.
- The Monitor commands `CALL -958` and `CALL -868` don't clear to end of display and end of line as they normally do; substitute (CONTROL)-(F) and (CONTROL)-(E).
- FLASH, INVERSE, and NORMAL don't work. Use (CONTROL)-(I) for INVERSE and (CONTROL)-(N) for NORMAL. There is no single-command FLASH equivalent.
- HRCG uses the DOS input/output hooks, so it can't be used directly with any other program that needs the hooks.

Memory Usage

HRCG and the character sets occupy the following amounts of memory:

- HRCG + standard character set: 2.0 K
- Each alternate character set: 768 bytes

Small Memory Apple II Users: The HRCG program will run on a 32K Apple II system if the second high-resolution graphics page isn't used; on such a small system the second page is the area of memory occupied by the Disk Operating System (DOS). Writing to the second page would clobber DOS. With the price of memory so cheap, however, it would be well worth your while to upgrade your system to at least 48K. Your Apple II uses 4116 dynamic RAM 200 nanosecond (or faster) chips; they are usually readily available from your Apple Computer dealer.

Entry Points and Technical Miscellanea

HRCG is an Rfile created by EDASM (see the *6502 Assembler/DOS Tool Kit Manual*, included with this package) and moved by the Relocating Loader to an appropriate place above HIMEM:.

As with other Rfiles (like APA, the subject of Chapter 1), HRCG must be relocated and initialized. After it is relocated by RLOAD (part of the Relocating Loader, covered in Chapter 7), it is initialized by a CALL to one of its two entry points. Since HRCG can be relocated to run at any address, entry points and other significant addresses are given in relocatable form relative to HRCG's first byte (called byte 0R—R for Relocatable). To find a particular absolute address, add the relocatable address to the relocation base returned from RLOAD:

Main entry point . . .0R
Secondary entry . . .3R

Below is a list of actions that occur when you issue a CALL to either of the entry points. Issuing a CALL command to the secondary entry makes the first two actions happen; a CALL to the main entry makes everything happen:

- HRCG is linked into the DOS input and output hooks (see your DOS manual)
- All parameters are set to their original values
- The display is set to full-display high-resolution graphics with the primary page showing
- The display is cleared and HRCG's title shows

Most of the technical information on HRCG really has to do with the Relocating Loader, covered next, in Chapter 7.

The Relocating Loader

The Relocating Loader

The Relocating Loader (Loader for short) lets you load special relocatable assembly language modules called **Rfiles** from an Applesoft program. Rfiles are created by the EDASM program included on the disk labeled DOS Programmer's Tool Kit Volume II, and instructions on how to create them (and why you'd want to) can be found in the *6502 Assembler/DOS Tool Kit Manual.* APA and HRCG are both Rfiles.

Rfiles are most commonly used by advanced assembly language programmers. But no matter how much or how little programming experience you might have, you need to know this material to use in your own programs the character sets you create with Animatrix.

Adding the Loader to Your Programs

The Relocating Loader consists of the two programs RBOOT and RLOAD, both to be found on the DOS Programmer's Tool Kit Volume I disk. Loader loads Rfiles just below HIMEM:, reducing HIMEM: by the length of the modules plus one byte. Here's how you'd use Loader in your own programs. The example assumes that the Rfile you're going to use already exists, and that RBOOT and RLOAD are in the appropriate drive.

What Code You Type	What Happens
1. `10 ADRS = 0`	Initializes a variable for the USR function
2. `20 PRINT CHR$(4);"BLOAD RBOOT"`	Loads RBOOT into memory; note use of `CHR$` function, and see warning note below
3. `30 CALL 520`	Activates RBOOT (which lives from 520 to 975), which in turn loads RLOAD

What Code You Type	What Happens
4. 40 ADRS = USR (0), "RFILENAME"	Loads named Rfile into memory at address returned to ADRS by the USR function (quotation marks are required around name)
5. 50 CALL ADRS	Activates RFILENAME

If you're fairly new to programming and aren't familiar with the more obscure aspects of linking machine language routines and Applesoft programs, use the above code exactly as it appears—except, of course, substitute the name of the Rfile you're using for the name RFILENAME. Use whatever line numbers you want; you can combine the statements onto one line if you're a fanatic byte-saver.

More often than not, other code will come between lines 40 and 50 (see MAXWELL, RIBBIT, SKYLAB, and especialy Chapter 8 of this manual for examples). There is, however, one textbook example on the disk that uses only the code given above—LOADAPA, lines 130-180.

▲ **Warning**

You must use the Loader before your program allocates any strings or dimensions any string arrays. If you set up strings before you use RLOAD, your strings will probably be destroyed and the entire structure of organized society as we know it will collapse.

Technical Details of Loader Operation

The Loader is actually made up of two machine-language programs, RBOOT and RLOAD. RBOOT loads RLOAD above the end of Applesoft's variable table and sets a USR function to point to RLOAD's entry point. Then RLOAD finds HIMEM:, loads the specified Rfile, and finally resets HIMEM: to just below the start of the Rfile. After all that, the USR function returns the starting address of the Rfile.

how line 40 works

Sophisticated Applesoft users and machine language programmers know that the USR function ordinarily looks quite different from line 40 in the above example. In fact, Applesoft would under most circumstances reject the whole line with a

syntax error or illegal quantity error, or the program would simply **hang**. Look at it again:

```
40 ADRS = USR (0), "RFILENAME"
```

In the normal flow of things, USR would pass its function—the number that appears in parentheses—to a machine language routine; the routine would then do something mathematical to the number and return the results to a variable. In this case the variable would be ADRS.

Here, the combination of RBOOT and RLOAD set memory up so that the ordinary flow of Applesoft is subverted. Applesoft reads line 40 up to the comma and does what you'd expect—it sends the number 0 down to the USR machine language program, which in fact was written by Loader. But then the ever-tricky Loader reads the rest of the line (the RFILENAME in quotes), BLOADs the specified Rfile just below HIMEM:, and then resets HIMEM: to just below the Rfile:

```
                  ①
40 ADRS = USR (0), "RFILENAME"④
       ②          ③
```

①Applesoft reads up to this comma first, goes to USR space

②HIMEM: returned into ADRS

③Loader makes Applesoft see a colon here instead of a comma

④Loader tells DOS to load specified Rfile at ADRS as if line read `BLOAD RFILENAME, A ADRS`

To see more examples of how to use the Loader in actual programs, see Chapter 8.

Chapter 8

Bringing It All Together

Bringing It All Together

Animatrix, HRCG, and the Relocating Loader taken together allow you to create and use your own character sets and graphics. The aim of this chapter is to clear up any confusion you may have about using the characters you create in your own programs.

This chapter begins with a listing of the code needed to use Animatrix-created character sets in any program. Next comes an analysis of how the experts who wrote MAXWELL and RIBBIT incorporate this code into their programs. Finally, there's a section on how the animation in MAXWELL works.

Before you read any further, get familiar with the animation programs on the DOS Programmer's Tool Kit Volume I disk from the user's point of view. Take a few minutes and run MAXWELL, SKYLAB, and RIBBIT. If you don't want to run all three, then just run MAXWELL; you'll be using it for demonstration purposes later. Then come back here.

What to Read First: This chapter may be difficult to understand if you haven't read the rest of this manual. Especially important are the chapters on Animatrix (Chapter 5) and High Resolution Character Generator (Chapter 6). Programmers with technical proclivities will want to read about the Relocating Loader (Chapter 7).

The Vital Code

The following routine represents the basis of all programs using Animatrix-created character sets. It is bare bones stuff with no error-trapping or REM statements, but it gets the job done.

Square brackets [ ] indicate where you are to supply required information. All other code is to be used exactly as shown. You can change the line numbers, but the sequence of lines should stay as-is:

```
100  ADRS = 0
110  PRINT CHR$ (4); "BLOAD RBOOT": CALL 520
120  ADRS = USR (0), "HRCG"

130 IF ADRS < 0 THEN ADRS = ADRS + 65536
140 CS = ADRS - 768 * [NUMBER OF SETS TO BE
    LOADED]: HIMEM: CS
150 CH = INT (CS / 256): CL = CS - CH * 256
160 POKE ADRS + 7, CL: POKE ADRS + 8, CH:
    CALL ADRS + 3

170 PRINT CHR$ (4); "BLOAD [1ST CHOSEN
    CHARACTER.SET], A"; CS
180 PRINT CHR$ (4); "BLOAD [2ND CHOSEN
    CHARACTER.SET], A"; CS + 768
190 PRINT CHR$ (4); "BLOAD [NTH CHOSEN
    CHARACTER.SET], A"; CS + 768 [*(N-1)]
200 PRINT CHR$ (4); "BLOAD [CHOSEN
    BACKGROUND], A$4000"
```

What the Code Means

Lines 100 through 120, plus the last statement on line 160, comprise the Relocating Loader, explained in Chapter 7.

Line 130 checks to see if ADRS's value, which represents an actual memory address, is expressed as a negative number. If it is, then the value gets changed to a positive number to make it easier to work with. Line 140 computes a new HIMEM: by looking at HIMEM:'s current location (stored in ADRS) and then subtracting from that number the space to be taken up by the character sets. The value is stored in CS (for Character Storage).

When you run your program later, HRCG is going to have to translate the regular ASCII characters in your program lines into the graphic characters you created using Animatrix. To know where to find the graphic characters, HRCG looks at locations 7 and 8 of its own code. Line 150's formulas convert the value of CS, which holds the location of the graphics character tables, into a form that can be stored in HRCG's 7th and 8th memory location via the POKE statements in line 160.

User-Supplied Information

Line 140 needs to know how many character sets you're going to use in your program so that HIMEM: can be located below them. Lines 170 through 190 want the names of the sets to load above HIMEM: so that your carefully designed character sets will be protected from strings, numeric variables, and other data.

Note that if you use more than one character set, each character set must follow the previous one consecutively in memory. Thus the second character set named in line 180 is to be loaded at HIMEM: + 768 (each character set is 768 bytes long). The formula supplied at the end of line 190 shows you how to handle the math for additional character sets.

Line 200 shows you how to load a background high-resolution picture onto page 2, which starts at memory location hexadecimal 4000 (decimal 16384).

About the Line Sequence: Lines 100 through 160 *must* appear in the order indicated, since the values and/or actions of each successive line depend upon the values computed or actions performed by the previous line. Lines 170 through 190 can appear anywhere after line 140, and line 200 can appear anyplace.

How the Experts Do It

It's one thing to isolate a programming technique out of context; it's quite another to actually use it. So that you can see how the experts use the code outlined above, you'll find the important programming lines from MAXWELL and SKYLAB below. You'll see that the two programmers implement the code with some minor differences from each other and from the sample code you saw earlier, but the essence is the same. Only the significant code is listed: a series of three dots (...) indicates that some commands or programming lines are purposely left out.

The MAXWELL code:

```
2020  ... : ADRS = 0
2030  PRINT CHR$ (4); "BLOAD RBOOT": CALL 520
2040  ADRS = USR (0),"HRCG"
            ...

2060  IF ADRS < 0 THEN ADRS = ADRS + 65536
2070  CS = ADRS - 768: HIMEM: CS
2080  D$ = CHR$ (4)
2090  PRINT D$; "BLOAD MAX.SET,A";CS
2100  CH = INT (CS / 256): CL = CS - CH * 256
2110  POKE ADRS + 7,CL: POKE ADRS + 8,CH: CALL
      ADRS + 3
2120  PRINT D$;"BLOAD APPLE LOGO.PIC,A$4000
```

The SKYLAB code:

```
2030  ADRS = 0 : ...
2040  PRINT CHR$ (4); "BLOAD RBOOT": CALL 520
2050  ADRS = USR (0),"HRCG"
            ...

2070  IF ADRS < 0 THEN ADRS = ADRS + 65536
2080  CS = ADRS - 3072: HIMEM: CS
2090  CH = INT (CS / 256): CL = CS - CH * 256
            ...


2110  PRINT CHR$ (4);"BLOAD SKY1.SET,A";CS
2120  PRINT CHR$ (4);"BLOAD SKY2.SET,A";CS +
      768
2130  PRINT CHR$ (4);"BLOAD SKY3.SET,A";CS +
      1536
2140  PRINT CHR$ (4);"BLOAD MUSH.SET,A";CS +
      2304
            ...
```

```
2160  PRINT CHR$ (4);"BLOAD WORLD
      MAP.PIC,A$4000"
           . . .

5030  POKE ADRS + 7,CL: POKE ADRS + 8,CH: CALL
      ADRS + 3
```

Animatrix/HRCG Program Lines

So far you've seen how HRCG and the Relocating Loader work together to store and activate Animatrix's character sets. Now with MAXWELL's help you'll learn how to access the special character sets in your own program lines.

You'll need Boston Window and APA in memory for this section. Type the following commands into your computer:

What You Type	What Happens
1. (CONTROL)-(RESET)	Sets up normal text screens
2. FP	Clears memory, resets HIMEM:
3. BRUN BOSTON WINDOW	Loads the Applesoft editor described in Chapter 2 (this takes a few seconds)
4. RUN LOADAPA	Loads special utility commands described in Chapter 1 (another brief delay while this program gets into memory)
5. LOAD MAXWELL	The program you want to look at
6. (CONTROL)-(E)	Enters the Applesoft editor
7. (CONTROL)-(G) 1000	Gets to the section of MAXWELL's code you want to inspect

the (CONTROL) key is labeled (CTRL) on the Apple II and II Plus

Your display looks like this:

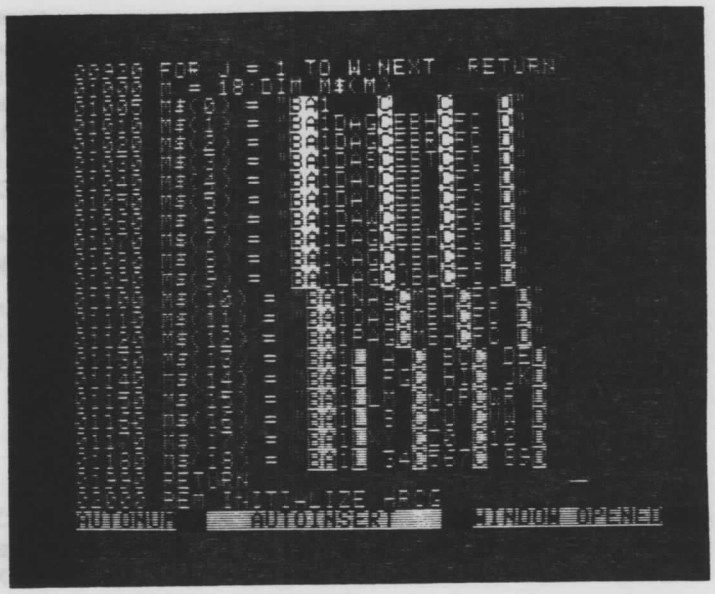
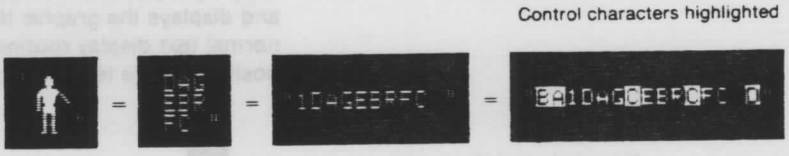


Figure 8-1. Section of MAXWELL Code Through Boston Window

In the section called Block Display—Soul of Character Animation in Chapter 6, you saw four ways the computer might treat line 1020 of MAXWELL:

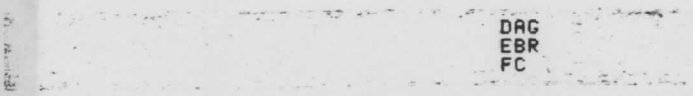
Figure 8-2. Different Ways Characters Can Appear



Lines 1005 through 1180 can be represented the same way; h these characters appear depends upon the filters through which you're looking. These few lines define all the shapes that Maxwell takes—standing, walking, waving his arms. It's the embedded control characters that give the character Maxwell his form when you (or more correctly, the computer) look at the code through HRCG.

How Your Apple Translates This Code

Because of the action of HRCG, line 1020 of this program looks like this on text page 1, the page you're used to looking at when you type in program lines:



HRCG does its stuff one character at a time. Take for example the letter A in the example. Uppercase A is the 34th character in the MAX.SET character table. When HRCG sees the A on its way to text page 1, HRCG goes to HIMEM: where the character sets are stored to see what graphic A represents. To find the 34th character HRCG multiplies 33 times 8 (it takes 8 bytes of high-resolution pixels to represent each plain old low-resolution text character; the first character starts at HIMEM:) and counts that many bytes into the table. There it finds the 8-byte high-resolution graphics character represented by the letter A. It takes that graphic, locates the proper position on the proper high-resolution graphics page and displays the graphic there. Then it passes the A onto the normal text display routine, which also places an A in the same position on the text display page (whew!).

See the *Applesoft BASIC Programmer's Reference Manual* for a discussion of text and high-resolution graphics display pages

Figure 8-3. MAXWELL's Head

A =

A Brief Diversion

Before going on to discover what makes Maxwell run, look at how MAXWELL's writer did his fancy text formatting. The writer embedded HRCG's Block and Text Overstrike commands into the code to underline the word `Maxwell` on the display.

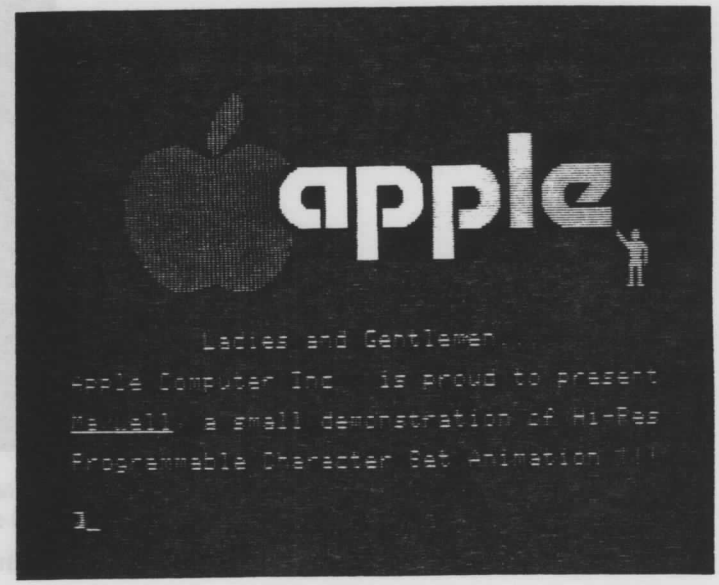
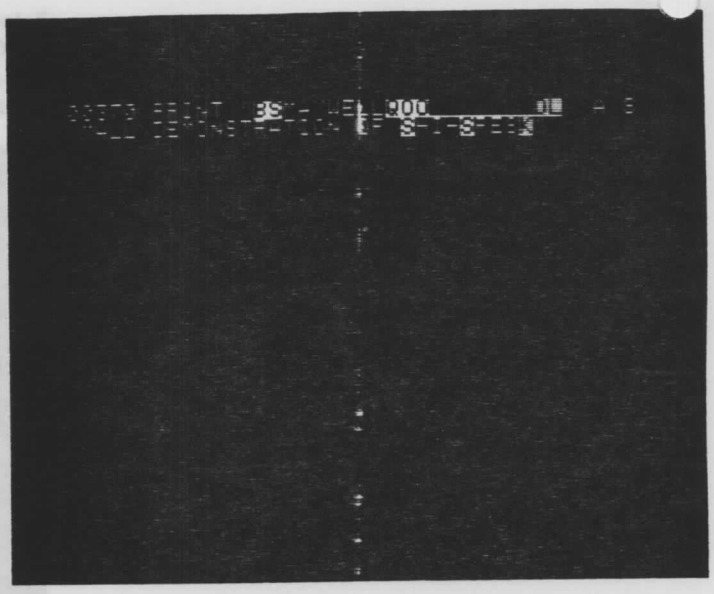


Figure 8-4. MAXWELL Screen at End of Program Run

Figure 8-5. Listing of Line 370



Translated into English, here's what the important section of line 370 says:

1. Start a Block ([CONTROL]-[B]).
2. Make the first character uppercase and then make the characters that follow lowercase ([CONTROL]-[S]): `Maxwell`.
3. Move the cursor to the upper left corner of the Block ([CONTROL]-[Q]) so that the next character to appear will be at the same position as the `M` in `Maxwell`.
4. Set up Overstrike ([CONTROL]-[O] [CONTROL]-[O]) so that the characters that follow will mix with the characters already on the screen rather than replace them.
5. Type underline characters so that `Maxwell` is underlined.
6. End the Block ([CONTROL]-[D]) and make the text that follows lowercase ([CONTROL]-[L]).

An Animation Technique

Lines 1005 through 1180 define all of Maxwell's different body positions. These lines also store those coded positions into an array called M$. Type FF to reset HIMEM: and run MAXWELL to initialize HRCG and to load the array. You can stop the program with a CONTROL-C if you don't want to see Maxwell go through his set of tricks again, but wait until you first see him walk onto the display before you do.

Maxwell's stroll across the screen is accomplished with just three graphics characters, or more accurately three blocks of characters. These blocks are stored in M$(16), M$(17), and M$(18).

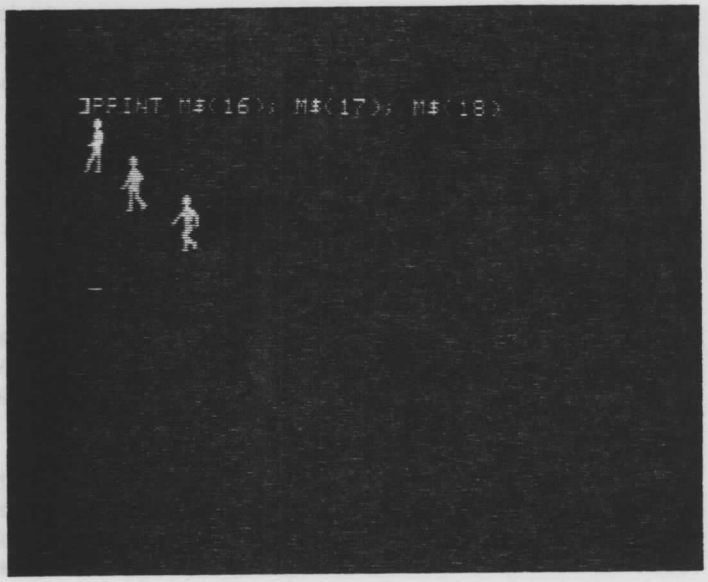


Figure 8-6. MAXWELL's Three Blocks of Characters

You'll be running MAXWELL a number of times, but only to see part of his antics. So that you don't have to watch the whole program again, add this line to MAXWELL:

```
155 STOP
```

and type the command RUN 50.

Line 50 is MAXWELL's soft entry point: it restarts the program without reloading programs from the disk. Watch Maxwell's movements carefully. Then add the following line to slow down his walk, and run the program from line 50 again:

```
145 SPEED= 50
```

Old Maxwell just shuffles his way across the screen, three different body positions (three elements of array M$) per shuffle. This action is accomplished in very few lines of code. Reset SPEED= to 255 and list lines 90 through 150 and 900 through 920.

These lines have Maxwell walking across the screen from right to left. In PRINT terms, a character block is printed in successive horizontal positions from HTAB 38 to HTAB 1 on the same vertical (VTAB 10) line:

```
①90   X = 38:Y = 10
  100  REM WALK ONSTAGE
②110  W = 60
③120  I = 16:  GOSUB 900
④130  I = 17:  GOSUB 900
⑤140  X = X - 1:I = 18:  GOSUB 900
⑥150  IF X > 1 THEN 120
      ...
  900  REM PRINT
⑦910  VTAB Y: HTAB X: PRINT M$(I);
⑧920  FOR J = I TO W: NEXT : RETURN
```

①These are values for horizontal and vertical screen positions

②For delay loop in line 920

③I will be an element of array M$ in the subroutine at 900

④Another element in the array

⑤Decreases horizontal position by 1, gets ready to use a third graphic block stored in array M$.

⑥If Maxwell isn't at the first position in the line yet, has him take another step

⑦Updates horizontal and vertical positions; displays the graphic

⑧A DELAY loop to slow Maxwell down, then a RETURN back to the main section

Each Maxwell shape is actually four characters wide and three display lines high. Before the horizontal position of the graphic block is advanced by 1 (line 140), Maxwell looks like he's taken several steps; and because Maxwell's fourth horizontal space is usually blank over all three vertical positions, he erases himself as he walks.

To see this a bit more clearly, you can make Maxwell run in place. First, press these command keys to reset HRCG's original parameters and to clear the screen:

CONTROL-Z CONTROL-P CONTROL-X

Now type the following line of direct execution code (no line numbers). It takes advantage of MAXWELL's subroutine starting at line 900, described earlier.

```
FOR S = 1 to 100: I = 16: GOSUB 900: I = 17: GOSUB
900: I = 18: GOSUB 900: NEXT
```

Experiment for a while with other sections of code in the program. Try changing some of the lines to see what happens. MAXWELL is an excellent program to play with since all its action occurs in such a short space of code. Be sure to look at MAXWELL's entire listing through the Boston Window editor, and see how the programmer uses HRCG's many commands.

A final word of caution, however: if you decide to list this program to a printer, you're in for a lot of blank sheets of paper! Lots of these lines have embedded in them HRCG's code for lowercase, CONTROL-L. Most printers see CONTROL-L as the ASCII code for form feed. Be sure to use Boston Window's REPLACE command to change those CONTROL-L's to something innocuous first.

Just for Fun Before You Leave...

Remove the extra line at 145 (you added it earlier to slow Maxwell down). To see Maxwell really zip across the screen, intercept line 920's DELAY loop by adding a line:

```
915 RETURN
```

and then type

```
RUN 50
```

Zip on, Maxwell!

Glossary of Terms

active character set: Character set HRCG is currently using to display graphics on either of the two high-resolution graphics pages.

active memory: Current Applesoft program memory area. Used to describe a ready-to-run program, versus a program or program segment stored above HIMEM: for later merging by APA.

APA: Abbreviation for Applesoft Programmer's Assistant.

binary file: A machine language program stored on a disk.

buffer: Temporary holding area for a string of characters.

clobber: To overwrite an area of memory usually reserved for code performing a primary function, thus destroying the primary function's code in the process.

DATA list: Items in a single DATA statement or series of DATA statements taken collectively.

default: The value or action selected by the system when the user does not select another expected or allowable value or ac... the standard condition.

display line: Actual or potential line of characters as it might appear on a display screen. A display line can be either 40 or 80 characters wide; up to 24 display lines can be visible at any one time. Contrast with **program line**.

display page: High-resolution graphics display area currently onscreen. The display page can be either graphics page 1 or page 2.

DOS: Abbreviation for Disk Operating System.

editor: Utility for creating, adding, or modifying lines in a computer program. In this manual, *editor* refers specifically to the Boston Window Applesoft editor described in Chapters 2 and 2e.

escape mode: Condition of computer, accessed by pressing ESC key, whereby the I, J, K, and M keys move the cursor up, left, right, and down the display without affecting the characters over which the cursor moves.

garbage: Assorted characters of no value to either human or machine strewn about the display or randomly scattered in a disk file.

hang: To be caught in a programming loop such that no useful work is performed. CONTROL-RESET returns control to the programmer.

highlighted text: Characters displayed in such a way that the normal foreground color becomes background and vice-versa; usually refers to black characters on a white or green background.

HRCG: Abbreviation for High Resolution Character Generator.

interleave: To merge program segments in such a way that the line numbers from one segment are interspersed with those of the second segment. For instance, if segment 1 had lines numbered 2, 4, and 6, and segment 2 had lines numbered 1, 3, and 5 before being merged, the interleaved program would have lines consecutively numbered 1 through 6. Each line would have its original number and its original text intact.

JSR: 6502 assembly language mnemonic for "jump to subroutine."

overlay: Condition whereby the dots making up a character already displayed are combined with a second character laid over the first. The effect is the same as typing one character, pressing the backspace key, and then typing a second character without the first character disappearing.

packed line: Program line with no spaces between key words.

page 0: The first 256 bytes in computer memory (hexadecimal locations $0 through $ff). Machine language programmers consider page 0 to be particularly valuable memory space because it takes only one byte of machine code to refer to any page 0 address.

page 1: The first of two high-resolution graphics display areas in the Apple's memory (decimal locations 8192 through 16383).

page 2: The second of two high-resolution graphics display areas in the Apple's memory (decimal locations 16384 through 24575).

primary page: High-resolution graphics display area (page 1 or page 2) to which any text or graphics information is written. The primary page is not necessarily the display page.

program editor: See **editor**.

program line: String of up to 239 characters preceded by a line number in the range 0 through 63999 and ending with a carriage return, consisting of one or more Applesoft statements.

pure cursor move: A move of the cursor having no effect upon the characters over which it passes.

Rfile: Relocatable file; machine language code written in such a manner that it can be located anywhere in memory and still operate properly.

screen line: See **display line**.

soft entry point: Starting point within a program such that not all of the program's usual functions are activated.

status line: Information displayed at the bottom of the screen during program line editing in Boston Window (Chapters 2 and 2e). The status line tells you whether the window is open or closed, if automatic line numbering is in effect, whether newly typed characters replace or are inserted among current characters, and so on.

technolalia: The speech of technotrolls.

technotroll: Technically oriented programmer, usually possessing sixteen fingers, who writes only in undocumented machine language.

utility: Program used as a tool in the development of other programs.

USR: A little-used Applesoft function whereby a machine language routine is called on by BASIC to do arithmetic or other operations upon a variable. The result is passed back to BASIC. See the *Applesoft BASIC Programmer's Reference Manual* for more details.

window: Frame surrounding a program line currently selected for editing in Boston Window (Chapters 2 and 2e). If the window is closed, cursor movement commands are valid only within the selected program line. Almost any editing command closes the window; CONTROL-O and certain other commands open the window to allow more general cursor movement.

Index

Cast of Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

S

T

U

V

W

X

Y

Z